Proc. VIIth Digital Image Computing: Techniques and Applications, Sun C., Talbot H., Ourselin S. and Adriaansen T. (Eds.), 10-12 Dec. 2003, Sydney

Computing Image-Based Reprojection Error on Graphics Hardware

John Bastian and Anton van den Hengel

The University of Adelaide, Adelaide, South Australia
Centre for Sensor Signal and Information Processing, Mawson Lakes, South Australia
john,anton@cs.adelaide.edu.au

Abstract. This paper describes a novel approach to the problem of recovering information from an image set by comparing the radiance of hypothesised point correspondences. This method is applicable to a number of problems in computer vision, but is explained particularly in terms of recovering geometry and camera parameters from image sets. The algorithm employs a cost-function to represent the probability that a hypothesised scene description and camera parameters generated the reference images and is characterised by its ability to execute on graphics hardware. Experiments show that minimisation of the cost-function converges to a valid solution provided there are adequate geometric constraints and projective coverage.

1 Introduction

Computing geometry in scene space is a relatively new approach towards computer vision problems that confers a number of advantages over feature-based algorithms. A number of these techniques rely on pixelwise comparison across an image set to recover information. The comparisons are usually carried out to find evidence to support or reject an hypothesised set of parameters, allowing conclusions to be drawn about the parameter space from repeated application of this process. The key to these approaches is pixelwise comparisons: a mathematically straightforward but computationally expensive process that requires computing occlusions and comparing projections of points. We present a method accelerating both these problems using commercially available graphics hardware.

One application of the graphics-based technique is towards the recovery of static geometry from a set of reference images. Solutions to this problem have, in general, focused on computing correspondences between features in the source images. These features are used to calibrate the cameras and compute a reconstruction. The quality of the camera calibration depends on the accuracy of these features. In contrast, the graphics-based approach compares synthetic images of a hypothetical scene with the reference images. There are a number of advantages to this approach, including the ability to recover scene shape without determining dense correspondences and its capacity to consider global structure when making local changes.

An example of this type of technique is space carving, which validates a hypothesised surface by considering the projection of each scene point into the reference images [1] [2] [3] [4]. The initial hypothesis is a conservative shape provided by the user guaranteed to encompass the reference geometry. A carving iteration validates every point by searching for projections that are inconsistent with the reference views. A point is considered valid if its projection fits a locally-computable light model in all images where it is visible; such points are said to be *photo-consistent*. A new hypothesised surface is created with inconsistent points removed. This changes the surface's visibility and requires further image-based comparisons until it converges to a state where no further points are removed.

Smelyansky *et al* use a similar basis to recover a height-map and make small changes to camera parameters from an initial feature-based estimate [5]. This approach considers entire images rather than determining photo-consistency for each point. Their algorithm, however, does not model occlusion, allowing their system to compute the derivative of the image with respect to parameter changes. New views are generated by applying the derivative to the last synthetic image. These synthetic images are compared with the reference images.

Both space-carving and Smelyansky's approaches treat reconstruction as a problem of colouring a hypothesised scene and comparing the results with the reference images. Computing the photo-consistency and generating new views involves finding the unoccluded projection in each image of a sub-set of the surface. Determining occlusion can be expensive, particularly if the scene structure is time-consuming to traverse or is not able to optimise the number of visibility tests. Searching detailed volumetric structures can be time-consuming because a large number of volume elements must be scanned to confirm that no opaque regions are closer to the camera than a given point.

1.1 Graphics hardware

Computing occlusion is a fundamental task in computer graphics that is efficiently addressed by graphics hardware. Modern Graphics Processing Units (GPUs) implement a pipeline organised in a series of transformations, clipping and rasterisation steps. Surfaces represented by triangles are sent to the pipeline as a stream of vertices where they are transformed and clipped by the vertex processor. The transformed triangles are rastered into a collection of *fragments*. This process linearly interpolates vertex attributes—such as texture coordinates computed by the vertex processor—and binds the new values to each fragment. The fragment processor uses these attributes to apply colour to the frame-buffer, which may involve referring to texture elements (*textels*) from the currently bound textures. The fragment processor is capable of performing operations on the fragment's attributes with the values sourced from the frame-buffer or other texture units.

This paper will describe a system that uses graphics hardware to test hypothesised parameters of the imaging process. These parameters are minimised by comparing the reprojection error between the reference images and synthetic images of the hypothetical scene. The synthetic images are computed on graphics hardware to leverage the inherent parallelism from an otherwise idle computational resource.

2 The photo-consistency cost function

Scenes that generate the reference images when imaged with the reference camera parameters are said to be photo-consistent, a property held by real, static scenes. The reprojection error is a measure of how closely the hypothetical scene recreates the reference images. This photo-consistency metric is used to reject hypothetical scene configurations, but this is an under constrained problem because there is an uncountably-infinite set of scenes can also be photo-consistent with a finite set of images [6].

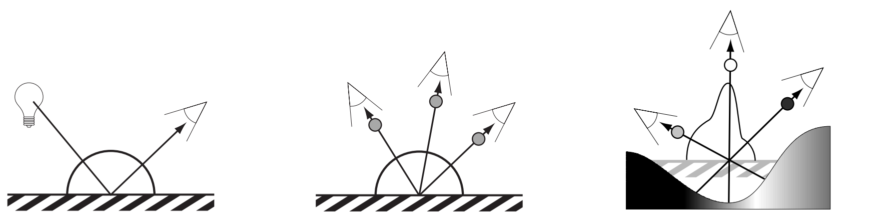
The diffuse model uniformly scatters light.
Observing uniform light supports the model.
Non-uniform light is not consistent with the model.

Fig. 1. Points on the surface can be assigned radiance parameters consistent with the reference images. Points not on the surface can appear to reflect light inconsistent with the model because the observed radiance is actually from different real scene points.

A point is photo-consistent if its light model can be assigned a parameter vector that model the point's projection in all reference views. This vector is usually a single colour to fit the diffuse light model, although more complicated models such as Phong [7] and Blinn [8] could be used. The only restriction on the choice of light model is that it must be *locally computable*; that is, the observed radiance must be independent of geometry elsewhere in the scene. This allows a consistency function to determine whether a point is consistent while ignoring global illumination properties such as reflection and refraction. It is not often possible for points that do not exist on the true surface to have suitable, globally-consistent radiance parameters because the back-projection of a point not on the true surface will actually map to different real scene points (see Figure 1). It is reasoned that such points are not part of the reference scene.

If we restrict the class of reference scenes to those that only contain diffuse light then the projection of a point in every image should be a similar colour. The probability that n images belong to this light model can be expressed by considering every pair images from that set and computing

$$\mathcal{O} = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \left(\sum_{p \in Vis(\mathbf{C}_i)} error(p, \mathbf{C}_i, \mathbf{C}_j, \mathcal{I}_i, \mathcal{I}_j) \right), \tag{1}$$

the reprojection error as a measure of how dissimilar a coloured hypothesised scene point is from the reference images. Here, $[\mathbf{C}, \mathcal{I}]$ and $[\mathbf{C}', \mathcal{I}']$ are two cameras and their reference images. We term these cameras the 'source' and 'destination' because the source camera is used to texture the scene that is then imaged by the destination camera.

The reprojection error is computed by texturing every scene point visible in the destination camera and comparing its projection in the destination reference image. The radiance observed by the destination camera $\mathbf{C}'$ with respect to the source $[\mathbf{C}, \mathcal{I}]$ is then

$$rad(\mathbf{p}, \mathbf{C}, \mathbf{C}', \mathcal{I}) = \mathcal{I}(\mathbf{C} \times closest(\mathbf{p}, \mathbf{C}')). \tag{2}$$

If we denote the camera's extrinsic matrix as$\mathbf{M}$, then the function

$$closest(\mathbf{p}, \mathbf{C}) = \underset{v \in projectionset(\mathbf{p}, \mathbf{C})}{\text{argmin}} \left\| \mathbf{M}v - \mathbf{M}v^{-1}\mathbf{0} \right\|, \tag{3}$$

computes the closest point to a camera from

$$projectionset(\mathbf{p}, \mathbf{C}) = \left\{ v \in \mathcal{V} \middle| \mathbf{C}v = \mathbf{p} \right\}, \tag{4}$$

the set of all scene-points that project to the pixel in the destination image. The synthetic radiance of a pixel is compared with its corresponding pixel in the destination image for photo-consistency by

$$error(\mathbf{p}, \mathbf{C}, \mathbf{C}', \mathcal{I}, \mathcal{I}') = consistency(rad(\mathbf{p}, \mathbf{C}, \mathbf{C}', \mathcal{I}), \mathcal{I}'(\mathbf{p})) \times occ(\mathbf{p}, \mathbf{C}, \mathbf{C}'). \tag{5}$$

The occlusion function weights the reprojection error for a pixel so only scene-points seen by both cameras are able to contribute to the error. This function is defined by

$$\begin{aligned} occ(\mathbf{p}, \mathbf{C}, \mathbf{C}') &= 1, \text{if } closest(\mathbf{p}, \mathbf{C}) = closest(\mathbf{p}, \mathbf{C}') \\ &= 0, \text{otherwise.} \end{aligned} \tag{6}$$

Under the diffuse light model, the likelihood that two points are photo-consistent is inversely proportional to the distance between the two colours

$$consistency(\mathbf{p}_i, \mathbf{p}_j) = \left\| \mathbf{p}_i - \mathbf{p}_j \right\|, \tag{7}$$

ie. the error is the distance between the colours in RGB space.

3 Computing photo-consistency on graphics hardware

Generating synthetic images from geometry described by triangles has received considerable attention from hardware vendors to the point where powerful commercially available graphics hardware—capable of processing millions of textured triangles per second—is relatively inexpensive [9]. Such graphics hardware can be leveraged to accelerate most of the cost-function by exploiting the inherent parallelism in graphics processing, caching data in the frame-buffer and using multiple texture references to combine the results of previous computation. The cost-function is implemented with three render passes: one pass to compute the occlusion of the scene in each camera and a final pass to compute the difference between the synthetic and real images.

3.1 Computing source camera occlusion

The first pass renders the scene from the source camera's view-point. This pass stores the set of points closest to the source camera in the depth buffer. Since only the depth buffer is important for this step, the colour buffers are disabled to decrease bandwidth across the video bus. The depth buffer represents the results of equation (3) for each point in the source camera and is cached in texture memory for later use. The colour buffers are then enabled and the depth buffer reset for the second pass.

3.2 Generating the surface texture

The second pass renders the scene from the destination camera's view-point. This pass implements equation (2) by generating a synthetic view of the scene with the texture mapped from the source image in a similar approach to Weinhaus *et al* [10]. It also computes equation 6 by storing the set of pixels used to compare with the destination's reference image.

The source image is back-projected onto the hypothesised scene using automatically generated texture coordinates. This is used to project the source camera's image onto the scene by using the source camera's projection matrix as the generation function. Texture is applied by linearly interpolating texture coordinates for each fragment within the triangle and copying the textel indexed by the texture coordinates to the frame-buffer. Textures mapped in this way are perspectively correct because texture coordinates are interpolated in homogeneous space.

Every triangle will have a valid projective texture because OpenGL's texture space is tiled [11]. This replicates the source image throughout the scene, attributing colours to scene points that are not visible by the source camera. These scene points will erroneously contribute to the cost when compared with the destination image if the incorrect assignments are left unchecked. Depth textures are used to track scene points occluded from the source camera by binding the depth buffer computed in the first render pass as a shadow map [12]. Figure

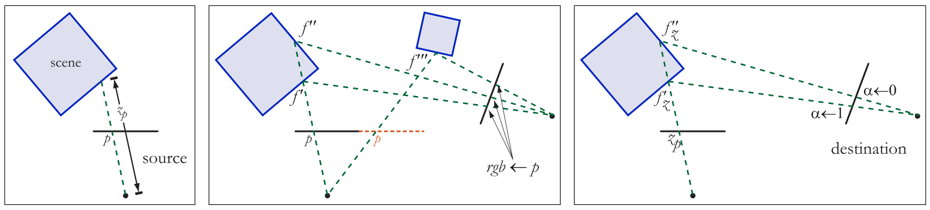

Fig. 2. The first pass computes the depth to each point visible by the source camera (i). The second pass textures the scene (ii). The third pass tests the depth of each fragment with the shadow map (iii).

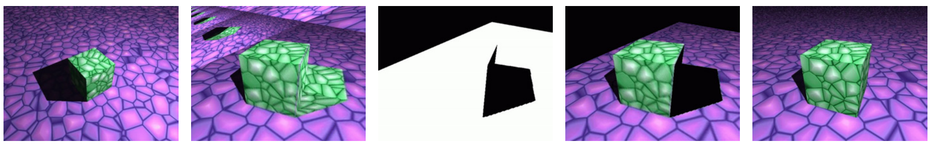

Fig. 3. Images generated by the cost-function for the second test with true parameters. The source image (i) is mapped to the scene (ii). Occluded pixels are stored in the α-channel (iii) and used to delete occluded pixels (iv). The synthetic view with true parameters closely resembles the destination image (v).

2 illustrates how the source image is propagated through the scene and how shadow maps are used to mask pixels occluded in the source camera.

Texture references to the shadow map are generated with the same function used to compute references to the source image. Instead of returning a coloured textel, shadow map references return α-textels as a function of the texture co-ordinates and the corresponding depth in the shadow map. Three-dimensional texture coordinates are computed for fragment: the first two elements are the point's projection in the source image and the third element is the distance between the fragment and the source camera. If the depth indexed by the first two coordinates is further than the depth to the source camera, then the fragment is not the closest point to the source camera. In this case, the shadow texture reference returns $\alpha \leftarrow 0$ to indicate the fragment is occluded. The textel $\alpha \leftarrow 1$ is returned if the test succeeds. An example of the texture scene and its shadow map is in Figure 3.

3.3 Comparing the synthetic and reference images

The reference destination image is composited with the synthetic image using the approach described by Duff [13] to implement equation 7. This is done using the blend operation $composite = \alpha s - \alpha d$ where s and d are the synthetic and destination pixels and α is taken from from the frame-buffer's α-channel. The final colour is the difference of the two colours if the pixel back projects to a scene-point visible by both cameras but is $[0,0,0,0]$ when the scene point is occluded in the source camera since $\alpha = 0$.

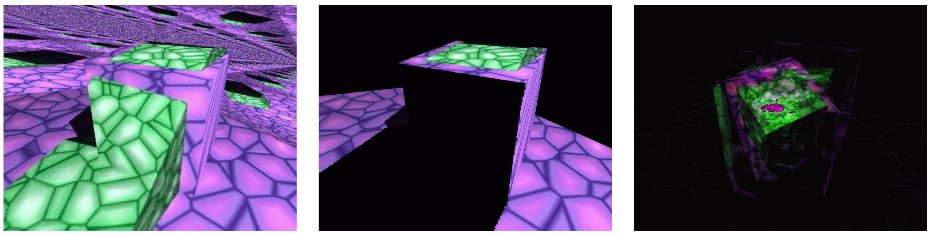

Fig. 4. Images generated by the cost-function for the second test with incorrect parameters. The source image is mapped to the scene (i) and occluded pixels are removed (ii). The result is stored in the accumulation buffer (iii).

The frame-buffer holds the colour disparity between two reference images with respect to the hypothesised scene. This result is stored in the accumulation buffer with weight $(n^2 - n)^{-1}$ [14]; an example of an accumulated buffer is in Figure 4(iii). The process from section 3.1 is repeated for every pair of images, each taking turns as a destination and source.

The final step is to compute the total reprojection error from each image pair. This is not easily computed on graphics hardware for three reasons: fragments cannot refer to adjacent pixels; the precision of colour channels is not sufficient; and OpenGL clamps pixels that exceed saturation. There are some techniques that could be used to partially accelerate the area-sum on hardware, including using floating-point buffers and 'shuffling' the frame-buffer through repeated rendering. Our approach, however, copies the frame-buffer to system memory and iterate over the pixels on the CPU.

4 Synthetic tests

The cost function was implemented with OpenGL 1.4 to test how it behaves when the imaging parameters are changed. Synthetic tests were used to provide ground truth as we are still evaluating whether the approach is valid. The input images were generated with the Persistence of Vision ray-tracer.

4.1 Estimating focal length

The first test was to determine if focal length could be estimated from synthetic images of a calbration grid when all other parameters known. Figure 5 graphs the relationship between the change in focal length and reprojection error. It demonstrates two important observations: the cost-function is minimised near the true focal length but is biased towards zero as the focal length increases.

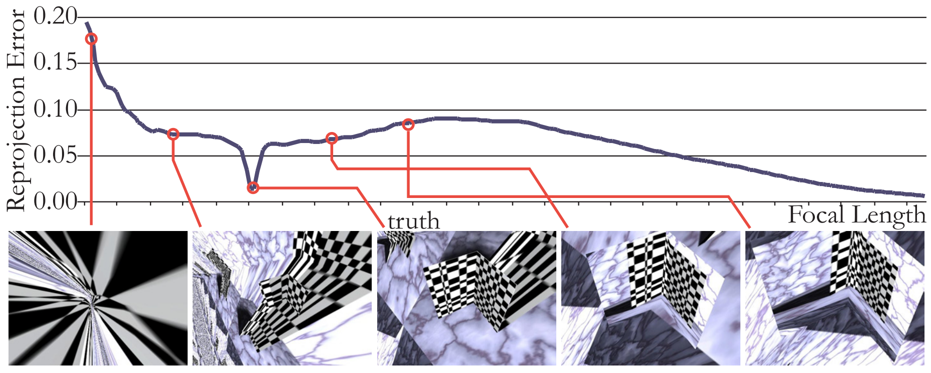

Fig. 5. The relationship between change in focal-length of a stereo-pair and the reprojection error

4.2 Estimating geometric parameters

The second test involved five cameras surrounding a cube upon a plane (see Figures 3 and 4) so that at least two cameras could see every side of the cube not on the ground plane. The cameras were considered calibrated, but the position, rotation and uniform scale of the box were left unknown.

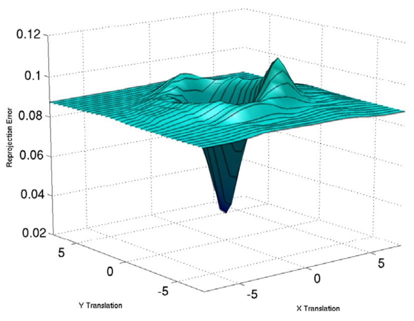

Fig. 6. The reprojection error as the cube's translation is varied. The true position is at [0, 0].

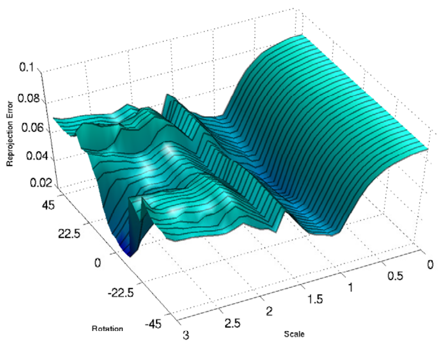

Fig. 7. The reprojection error as the cube's rotation and scale parameters are varied. The true parameters are 0(90) degrees rotation and 1.0 scale.

Figure 6 graphs the cost-function when the scale and rotation are fixed at their true value but the position of the cube is varied. The cost is minimised near the true position and approaches a constant value as the box is moved. The constant value is a result of pushing the box out of the camera's visual hull where the error is from the projection of the source images onto the static ground plane.

Figure 7 graphs the cost-function with the position fixed at the true location but with varying rotation and scale parameters. The graph near the true parameters is characterised by an apparent dominance in change of scale over rotation, but this is because scale has a much greater impact on the error while the cube's projection in each camera is relatively small. The impact is small because the silhouette of a cube remains relatively constant from different viewing-angles; we would expect an increased change in error if the shape was more complicated. The cost-function approaches a minimum in two key areas: the first minimum is at the true parameters, found within the valley where scale=1.0; and the second is when the rotation approaches 0 as the scale increases. This minimum reflects a bias against occlusion.

5 The occlusion bias

The cost-function biases occlusion because it relies on comparing scene points visible to a pair of cameras. The reprojection error is consequently minimised if there are no visible scene points to contribute to the error, leading to a bias that pushes the parameters so no geometry is visible. This problem is apparent in both tests. Increasing the focal length reduces the field of view of each camera to the point where each camera projects its image onto a small part of the scene not visible by the other camera. Similarly, each camera's projection on the scene is reduced when the cube's scale is increased. Useful geometric parameters can be found if appropriate priors to constrain the parameter space. The focal length test would benefit from this by capping the focal length to ranges that generate sensible field-of-views.

The visibility bias cannot be overcome by penalising occluded pixels. If the cost-function is modified to include a default penalty for occluded pixels, then problem becomes one of deciding what penalty to apply. If the bias is too low then the cost function will bias occlusion (its current behaviour, since the weight is effectively 0); however, if it is too high then it will tolerate significant inconsistency to minimise occlusion. The ideal solution would involve an *a priori* estimation how much geometry is visible in pairs of cameras.

6 Conclusion and Future Work

There are a number of advantages to using a graphics-based approach towards recovering scene information. The key advantage is that parameter changes has a global consequence to the shape's reprojection. These approaches shift the problem from image-processing into one where changes are made in scene-space so they are consistent when imaged with the reference cameras. The key to these approaches is comparing an hypothesis with the images which involves comparing unoccluded scene points with the images. Our goal was to accelerate this comparison in graphics hardware. We re-parameterised the colour consistency cost function so it can be mapped on graphics hardware. This cost function shows that it is minimised near the true parameters provided parameters are clamped to prior ranges and a sufficient number of cameras to reduce the visibility bias.

The difficulty with the reconstruction framework is its reliance on the parameterised scene model provided by the user. The initial scene graph is constructed with a geometry modeller that allows the user to define constraints on parameter ranges. It is planned for future work to use a hierarchical scene model to alleviate both the demands on the user to provide the initial geometry and minimise the number of free parameters. This system would optimises a coarse model and then refine the estimate by breaking polygons with a high reprojection error, loosen the scene parameters and re-optimise. An example of this is recovering sides of buildings that are not flat from brick extrusions. It is hope the inital cube model would locate the general area of the building and successive partitioning would resolve the bricks.

References

1. Culbertson, W.B., Malzbender, T., Slabaugh, G.G.: Generalized voxel coloring. In: Workshop on Vision Algorithms. (1999) 100–115
2. Seitz, S., Dyer, C.: Photorealistic scene reconstruction by voxel coloring. Proc. CVPR (1997) 1067–1073
3. Kutulakos, K.N., Seitz, S.M.: What do photographs tell us about 3d shape? Technical Report TR692, Computer Science Dept., U. Rochester (1998)
4. Kutulakos, K.N., Seitz, S.M.: A theory of shape by space carving. Technical Report TR692 Computer Science Dept., U. Rochester (1998)
5. Smelyansky, V.N., Morris, R.D., Kuehnel, F.O., Maluf, D.A., Cheeseman, P.: Dramatic improvements to feature based stereo. In: Proc. European Conference on Computer Vision. (2002)
6. Poggio, T., Torre, V., Koch, C.: Computational vision and regularization theory. Nature **317(26)** (1985) 314–319
7. Phong, B.: Illumination for computer generated images. Communications of the ACM **18** (1975) 311–317
8. Blinn, J.F.: Models of light reflection for computer synthesized pictures. ACM Computer Graphics SIGGRAPH **19(10)** (1977) 542–547
9. Thompson, C.J., Hahn, S., Oskin, M.: Using modern graphics architectures for general-purpose computing: a framework and analysis. In: Proc. Computer Graphics and Interactive Techniques. (2002)
10. Weinhaus, F.M., Devarajan, V.: Texture mapping 3d models of real-world scenes. ACM Computing Surveys **29(4)** (1997) 325–365
11. Segal, M., Akeley, K.: The OpenGL graphics interface. Technical report, Mountain View, CA,USA (1993)
12. Segal, M., Korobkin, C., Foran, J., Haeberli, P.: Fast shadows and light effects using texture mapping. In: Proc. Computer Graphics and Interactive Techniques. (1992)
13. Porter, T., Duff, T.: Compositing digital images. In: Proc. Computer Graphics and Interactive Techniques. (1984)
14. Haeberli, P., Akeley, K.: The accumulation buffer: hardware support for high-quality rendering. In: Proc. Computer Graphics and Interactive Techniques. (1990)