The Real-Time Reprojection Cache

Diego Nehab[1] Pedro V. Sander[2] John R. Isidoro[2]
[1]Princeton University [2]ATI Research

Abstract

We describe a simple and inexpensive method that uses stock graphics hardware to cache and track surface information through time. Cached information is stored in frame-buffers, thereby avoiding complex data-structures and bus traffic. When a new frame is rendered, an efficient reprojection method gives each new pixel access to information computed during previous frames.

This idea can be used to adapt a variety of real-time rendering techniques to efficiently exploit spatio-temporal coherence. When applications are pixel bound, the cached algorithms show significant cost and/or quality improvements over their plain counterparts, at virtually no extra implementation overhead.

1 Exposition

Most real-time rendering applications and multi-pass rendering algorithms exhibit a considerable amount of spatio-temporal coherence (see examples in figure 1). This coherence can be exploited if, in the process of computing a new frame, we can efficiently access surface properties computed in previous frames.

Although reprojection has been extensively studied in the past, previous work focuses on offline renderers, such as ray-tracers [Bala et al. 1999; Tole et al. 2002], describes custom hardware architectures designed to exploit coherence [Regan and Pose 1994; Torborg and Kajiya 1996], or is related to image-based rendering techniques [McMillan and Bishop 1995; Mark et al. 1997]. In contrast, we present an implementation that maps naturally to current graphics hardware and real-time rendering applications.

Because of spatio-temporal coherence, currently visible surface points were probably also visible in the previous frame. This justifies the policy of only caching information for visible pixels (such as final color, shadow map tests etc). When a new frame is rendered, this information can be obtained by texture-fetching into previously rendered frame-buffers. For that, we need to compute the appropriate reprojected texture coordinates and determine if each pixel was indeed visible in the previous frame.

Reprojected texture coordinates can be efficiently obtained with standard hardware support. While rendering a frame, we associate to each vertex v the homogeneous projection space coordinates (x_v, y_v, z_v, w_v) it had when the previous frame was rendered. Automatic perspective correct interpolation of these coordinates give each current pixel p access to the previous coordinates (x_p, y_p, z_p, w_p) of the generating surface point. Final texture coordinates are then obtained by division by w_p within the pixel shader.

To determine whether a surface point that is currently visible was also visible in the past, we check its reprojected depth against the depth stored in the previous frame-buffer. If the reprojected depth is within ε of the stored depth, we have a cache-hit. Otherwise, we have a miss. Robustness is achieved by using bilinear interpolation while reading the stored depth and by setting ε to the smallest z-buffer increment.

The application can now run different pixel shaders for the cache-hit and cache-miss cases. On a hit, a cheap shader can use cached information directly, or combine it with new incremental computations. On a miss, an elaborate shader can compute all necessary values from scratch. Whenever there is enough spatio-temporal coherence, there will be far more hits than misses. Our tests indicate that hit ratios of 90% and up are common. This can lead to substantial performance and/or quality improvements.

Figure 1: *Cache-hits and misses. Green regions show pixels that were visible in previous frames. Red pixels show newly visible pixels. Results were extracted from animation sequences and produced by our algorithm in real-time.*

2 Results and conclusions

Each application we investigated was either authentically pixel bound, or made so by use of an expensive (albeit not unreasonable) pixel shader. In stereoscopic rendering, depth-of-field and motion blur, the cached multi-pass algorithms obtained speedups from 30 to 100% with regard to the brute force multi-pass method. In shadow mapping, the RRC allowed us to effectively use four times more shadow map samples per pixel, leading to substantially better quality at no performance cost.

The RRC works well in animation sequences, although it requires twice the amount of vertex processing. As long as the application is pixel bound (like most modern games), this shouldn't be an issue. The method doesn't apply to transparent objects, but those can be trivially tagged and considered cache misses. Dynamic properties can be handled by a simple aging mechanism.

References

BALA, K., DORSEY, J., and TELLER, S. 1999. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics*, 18(3):213–256.

MARK, W. R., MCMILLAN, L., and BISHOP, G. 1997. Post-rendering 3D warping. In *Symposium on Interactive 3D Graphics*, pages 7–16.

MCMILLAN, L. and BISHOP, G. 1995. Head-tracked stereoscopic display using image warping. In S. Fisher, J. Merritt, and B. Bolas, editors, *SPIE*, volume 2049, pages 21–30.

REGAN, M. and POSE, R. 1994. Priority rendering with a virtual reality address recalculation pipeline. In *Proc. of ACM SIGGRAPH 94*, ACM Press/ACM SIGGRAPH, pages 155–162.

TOLE, P., PELLACINI, F., WALTER, B., and GREENBERG, D. P. 2002. Interactive global illumination in dynamic scenes. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2002)*, 21 (3):537–546.

TORBORG, J. and KAJIYA, J. T. 1996. Talisman: commodity realtime 3D graphics for the PC. In *Proc. of ACM SIGGRAPH 96*, ACM Press/ACM SIGGRAPH, pages 353–363.