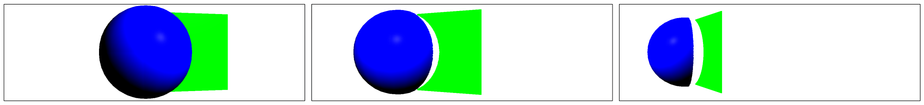
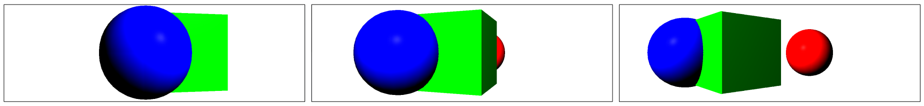
Reducing Geometry-Processing Overhead for Novel Viewpoint Creation

Francisco Inácio Jan P. Springer

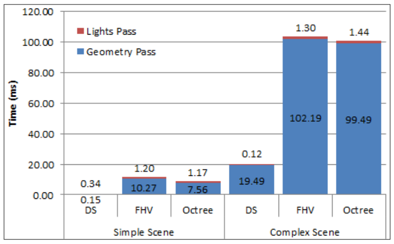
University of Hull



(a) *Deferred shading, 0° offset.* **(b)** *Deferred shading, 30° offset.* **(c)** *Deferred shading, 60° offset.*



(d) *FHV rendering, 0° offset.* **(e)** *FHV rendering, 30° offset.* **(f)** *FHV rendering, 60° offset.*



(g) *Performance comparison.*

Figure 1: *Screen shots (a) and (d) show a scene rendered using deferred shading and fragment-history volumes, respectively. (b) and (c) reuse the existing frame-buffer data to reconstruct for new viewpoints using deferred shading while (e) and (f) reconstruct the scene using fragment-history volumes for the same set of viewpoints. (e) and (f) correctly reconstruct the scene for the viewpoint changes but (b) and (c) exhibit artifacts and cannot reveal previously hidden objects. (g) provides an overview of initial performance data.*

1 Introduction and Motivation

Maintaining a high steady frame rate is an important aspect in interactive real-time graphics. It is mainly influenced by the number of objects and the number of lights to be processed for a 3d scene. The upper-bound effort for rendering a scene is then defined by the number of objects times the number of lights, i. e. $\mathcal{O}(N_O \cdot N_L)$. Deferred shading reduces this upper bound to the number of objects plus the number of lights, i. e. $\mathcal{O}(N_O + N_L)$, by separating the rendering process into two phases: geometry processing and lighting evaluation. The geometry processing rasterizes all objects but only retains visible fragments in a G-Buffer for the current viewpoint. The lighting evaluation then only needs to process those surviving fragments to compute the final image (for the current viewpoint). Unfortunately, this approach not only trades computational effort for memory but also requires the re-creation of the G-Buffer every time the viewpoint changes. Additionally, transparent objects cannot be encoded into a G-Buffer and must be separately processed. Post-rendering 3d warping [Mark et al. 1997] is one particular technique that allows to create images from G-Buffer information for new viewpoints. However, this only works with sufficient fragment information. Objects not encoded in the G-Buffer, because they were not visible from the original viewpoint, will create visual artifacts at discontinuities between objects. We propose fragment-history volumes (FHV) to create novel viewpoints from a discrete representation of the entire scene using current graphics hardware and present an initial performance comparison.

2 Our Approach

FHVs are based on the idea of the A-Buffer and can be implemented on current graphics hardware as per-pixel linked lists of fragments [Yang et al. 2010]. Our prototype software uses shader-storage buffer objects in OpenGL similar to [Crassin 2010]. However, FHVs are biased towards the direction of the original viewpoint, i. e. the fragment density along the original view direction is much higher than for any orthogonal direction. Using simple point splatting to visualize the captured fragments exhibits artifacts for novel viewpoints. To alleviate this we render the scene from each of the main directions, i. e. X, Y, and Z, and store the resulting fragments into an octree structure. Instead, of using a per-pixel linked lists of fragments we use per-octant linked lists of fragments.

3 Initial Results

We compare the performance of the geometry-processing stage for deferred shading (DS), FHVs using a per-pixel linked list of fragments (FHV$_{\text{G}}$), and FHVs using an octree representation of fragments from one basis direction (FHV$_{\text{OCT}}$). We used two different scenes for our tests: a simple low depth complexity scene and a scene with high polygon count as well as high depth complexity (cf. figure 1g). Our tests indicate that, for DS with the simple scene, the geometry pass took 0.15 ms and the lighting pass took 0.34 ms on average. The correspondent values for FHV$_{\text{G}}$ and FHV$_{\text{OCT}}$ were 10.27/1.20 ms and 7.56/1.17 ms, respectively. For the complex scene DS took 19.49 ms for the geometry pass and 0.12 ms for the lighting pass. The correspondent values for FHV$_{\text{G}}$ and FHV$_{\text{OCT}}$ were 102.19/1.30 ms and 99.49/1.44 ms, respectively.

For static viewpoints DS clearly outperforms our FHV technique, i. e. the DS lighting pass is, on average, 10 times faster than either that of FHV$_{\text{G}}$ or FHV$_{\text{OCT}}$. However, in the case of a (constantly) changing viewpoint DS needs to execute its geometry pass each frame while our FHV technique simply reuses the spatial representation of the scene. For the simple scene DS is still faster due to less overhead by either storing a fragment's data or simply discarding it after the rasterization stage. However, for complex scenes the DS's geometry pass becomes the bottleneck. In contrast, our FHV technique only needs to execute the lighting pass. Frame times are then: DS 19.49 ms + 0.12 ms vs. FHV$_{\text{OCT}}$ 1.44 ms.

4 Next Steps

Currently we only use a simple point-splatting technique to evaluate the fragments in the lighting pass. FHVs storing fragments in an octree will allow for lighting evaluation based on ray casting. This will also allow for correct transparency blending, shadow computation, and indirect lighting, to name just a few techniques. While FHVs are able to create novel viewpoints from static scene data they would incur a large overhead for scenes with dynamic elements. However, FHVs based on an octree structure would allow for incremental updates of specific objects, probably best in combination with compute pass(es) on the GPU.

References

CRASSIN, C. 2010. OpenGL 4.0+ ABuffer V2.0: Linked lists of fragment pages. last accessed: 2015-05-13, `http://blog.icare3d.org/2010/07/opengl-40-abuffer-v20-linked-lists-of.html`.

MARK, W. R., MCMILLAN, L., AND BISHOP, G. 1997. Post-Rendering 3D Warping. In *I3D '97: Proceedings of the 1997 Symposium on Interactive 3D Graphics*, ACM, 7–16.

YANG, J. C., HENSLEY, J., GRÜN, H., AND THIBIEROZ, N. 2010. Real-Time Concurrent Linked List Construction on the GPU. *Comput. Graph. Forum 29*, 4, 1297–1304.

SIGGRAPH 2015 Posters, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3632-1/15/08.
http://dx.doi.org/10.1145/2787626.2792599