Z-fighting aware Depth Peeling

Andreas A. Vasilakis*
University of Ioannina

Ioannis Fudos†
University of Ioannina

1 Introduction

Efficient capturing of the entire topological and geometric information of a 3D scene is an important feature in many graphics applications for rendering multi-fragment effects. Example applications include order independent transparency, volume rendering, CSG rendering, trimming, and shadow mapping all of which require operations on more than one fragment per pixel location.

An influential multi-pass technique is *front-to-back* (F2B) depth peeling [Everitt 2001] which works by peeling off a single fragment per pass and by exploiting the GPU capabilities to accumulate the final result. The major drawback of this peeling algorithm is that fragment layers with depth identical to the fragment depth detected in the previous pass are discarded and so not peeled. *Stencil Routed A-buffer* (SRAB) [Myers and Bavoil 2007] treats z-fighting for sorted fragments. However, SRAB is limited by the resolution of the stencil buffer and is incompatible with hardware supported multisample antialiasing. *k-buffer* [Bavoil et al. 2007] processes k fragments in a single pass, thus performing up to k times faster than F2B. k-buffer suffers from read-modify-write hazards and needs a small fixed amount of additional memory which is allocated in the form of multi render target buffers. Similarly to SRAB, k-buffer requires a pre-sorting of the primitives of the scene to treat correctly up to k Z-fighting fragments.

In this work, we introduce a novel technique for commodity graphics hardware that completely treats Z-fighting by extending F2B depth peeling with the overhead of one extra geometry pass. To speed up depth peeling at scenes with large number of layers with same depth values, we also propose an approximate z-fighting free depth peeling technique that combines the F2B and the k-buffer algorithms.

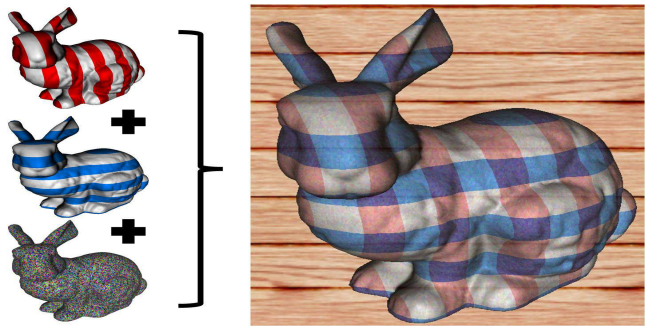

Figure 1: *Order independent transparency with z-fighting corrections for a scene consisting of 3 bunnies.*

2 Z-fighting Correction Algorithms

Extending F2B: We present the Z-fighting free F2B (*F2B_ZF*) depth peeling by adapting the F2B algorithm to peel all fragments placed at the same depth. When we have peeled all fragments at one depth, the next depth layer underneath is returned. To obtain all fragments at the same depth we discard the fragments that are not placed at this depth. To distinguish among z-fighting fragments, we extract (peel) the color of the fragment which has the maximum primitive identifier (build-in variable $gl\_PrimitiveID$ of GLSL [Kessenich 2009]). One extra geometry pass is used to calculate the sum of the (remaining, not peeled) z-fighting fragments and find which of them should be extracted next. We discard peeled fragments of the same depth by eliminating all fragments that have a primitive identifier equal or larger than the maximum primitive id determined during the previous step. Both computations are performed in one pass using additive and maximum blending operations respectively. The overhead of this algorithm as compared to the original F2B is the increase of the number of geometry passes from a scene with depth complexity N to $2N + 1$. The plus one pass overhead is due to the fact that the first depth peeling pass serves only as an initialization of the depth (no color information is extracted):

Algorithm 1 F2B_ZF Depth Peeling

{1st Rendering Pass using Max Blending}
1: **if** not first pass **then**
2: **return** The color of the fragment with the maximum ID
3: **end if**
4: **if** all fragments at this depth have been peeled **then**
5: **return** next nearest layer depth
6: **else**
7: **return** same layer depth
8: **end if**

{2nd Rendering Pass using Add and Max Blending}
1: **for all** not peeled fragments placed at current depth layer **do**
2: **return** The sum of them
3: **return** The maximum ID of them
4: **end for**

Combining F2B with k-buffer: We further introduce *F2BKB_ZF*, a depth peeling technique combining traditional depth peeling with the k-buffer approach. At each iteration, a depth layer is extracted using the classical F2B rendering pass. Then, using a variation of k-buffer we only extract fragments positioned at the same depth layer. Note that initial primitive sorting is not needed since we peel fragments at the same depth. While this method is much faster for many scenes, it cannot eliminate the z-fighting effect when there are more than k fragments at the same depth.

3 Results

Methods	MB	Sorting	MSAA	Peeled Layers	Z-fighting Accuracy	Total Passes	FPS
F2B	9	×	√	[8,8,8,8]	[100,25,12.5,0.08]	[8,8,8,8]	[219,83,45,9]
k-buffer	102	√	√	[8,29,56,56]	[100,90,87.5,58.3]	[1,4,7,7]	[147,23,8,4]
SRAB	126	√	×	[8,32,48,48]	[100,100,75,50]	[1,4,6,6]	[123,19,10,7]
F2B_ZF	33	×	√	[8,32,64,96]	[100,100,100,100]	[17,65,129,193]	[125,14,4,1]
F2BKB_ZF	33	×	√	[8,32,64,64]	[100,100,100,66.6]	[16,16,16,16]	[59,30,11,5]

Table 1: *Comparison between traditional depth peeling techniques and proposed methods.*

Figure 1 illustrates transparent rendering for three differently rendered Stanford Bunnies (69,451 triangles) using z-fighting correction. Finally, Table 1 shows a comparison in terms of peeling accuracy, performance (in fps) and memory storage (in Mbytes) of the original F2B, k-buffer and SRAB methods and both of our proposed alternatives for a scene consisting of $[1, 4, 8, 12]$ bunnies at a 1024×768 viewport on an nVidia Geforce GTX 480. Finally, the software is available at `http://www.cs.uoi.gr/~fudos/siggraph2011.html`.

References

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, J. A. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the GPU using the k -buffer. *Proceedings of the 2007 symposium on Interactive 3D graphics and games - I3D '07*.

EVERITT, C., 2001. Interactive Order-Independent Transparency, Tech. Report, Nvidia Corporation.

KESSENICH, J., 2009. The opengl shading language version: 1.50, document revision: 11.

MYERS, K., AND BAVOIL, L. 2007. Stencil Routed A-Buffer. *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*.

*e-mail:abasilak@cs.uoi.gr
†e-mail:fudos@cs.uoi.gr