GUARDED ORDER INDEPENDENT TRANSPARENCY

Lucian PETRESCU[1], Florica MOLDOVEANU[2], Anca MORAR[3], Victor ASAVEI[4], Alin MOLDOVEANU[5]

Order independent transparency represents a class of graphics algorithms in which the final result is independent of the order of primitive rasterization. The complexity of this class is given by the fact that the fragments generated by the rasterized geometry need to be sorted and stored on a per pixel basis - a computational and a memory bandwidth problem. In this paper we present a novel real time single pass algorithm that uses fragment culling in order to decrease the memory usage by an average of 12.5% while still performing virtually correct transparency.

Keywords : computer graphics, order independent transparency, culling

1. Introduction

Order independent transparency (OIT) algorithms are necessary to correctly render transparent geometry in rasterization based computer graphics. The first technique used for OIT was object ordering as proposed by Porter and Duff[1]. However, this is unable to correctly render interpenetrating primitives like Mobius strips because any pre-sorting stage does not sort at a per-fragment level. Thus, correct transparency can only be solved during rendering.

The only way in which the transparency problem can be solved is saving all of the samples which then, composed together, form the final image. One important property of this approach is that the scene is divided in a number of layers, as pointed out by Everitt [2]. This number is defined as the number of intersections between ray that goes from the camera and through the pixel and the scene.

[1] Msc. eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: alexandru.petrescu@cti.pub.ro
[2] Prof., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: florica.moldoveanu@cs.pub.ro
[3] PhD eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: anca.morar@cs.pub.ro
[4] PhD eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: victor.asavei@cs.pub.ro
[5] PhD eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: alin.moldoveanu@cs.pub.ro

Depth peeling [2] and dual depth peeling [3] are algorithms which render one, respectively, two layers at a time and the composition is done incrementally. The main problem of this approach is that there must be a total of N, respectively, N/2 rendering passes. This solution is not practical for very complex scenes.

A fixed number of fragments can be pre-allocated for each pixel, in a memory area filled by rendering [4]. The difficulty with this method is that it can allocate more than it is required for some pixels and allocate less that it is necessary for others. This behavior stems from the fact that the number of layers is a per-pixel property and not a general one, therefore generalization is very inefficient.

A dynamic number of fragments per pixels can be used [4], but such an approach is limited to Shader Model 5 hardware. A large area of memory is allocated per frame and each pixel writes to this area depending on necessity, generating a linked list of samples which will then be sorted and composited. In this way pixels with many layers will have sufficient memory space to be saved and pixels with very few layers will not waste memory. On the other hand this technique is very expensive because memory bandwidth is limited on the GPU.

The size of the buffer used can be computed on a per frame basis [5] but this will require a second rendering pass in the algorithm, which, depending on the vertex density of the scene, can lead to performance problems.

Both the per pixel lists method and the pixel pre-allocated memory method can be improved from a performance point of view with stochastic algorithms [6] that try to compress information when the provided memory budget is not sufficient. This is done through methods which define data importance by different criteria such as visibility functions. On the other hand statistical methods are not exact and are computationally much more expensive than the other algorithms.

2. Background

In order to obtain the composite value of a pixel (P_i) each fragment intersected by the ray from the eye to P_i is considered as a sample (S_i) and all the samples are used with a combination function F. This function may or may not require visibility [7]. Visibility is defined as a function which is the total transmittance between the eye and the sample. The contribution of a sample with $vis(z_i)$ is:

$$Contrib(z_i) = C_i \cdot A_i \cdot vis(z_i) \tag{1}$$

Where z_i is the distance from the viewer, c_i is the color of the sample and a_i is the transparency of the sample. The total contribution of all samples on a given ray is:

$$C(P_i) = \sum_{i=0}^{n} c_i \cdot A_i \cdot vis(z_i) = \sum_{i=0}^{n} Contrib(z_i) \tag{2}$$

If the visibility function is not to be computed per sample, it can be bypassed by sorting the samples and iteratively merging them, thus implicitly computing each visibility function for each sample. The equations for sample contribution were first given by Porter and Duff [1] and are known as the blending equations. In the back to front composition a new sample reduces the visibility of the current accumulated color in that pixel:

$$C'_{acc} = (1 - a_i) \cdot C_{acc} + a_i \cdot C_i \tag{3}$$

In the front to back equations a new fragment has its visibility reduced by the accumulated color and visibility:

$$C'_{acc} = C_{acc} + a_{acc} \cdot a_i \cdot C_i \tag{4}$$

Fig. 1. The Dabrovic Sponza scene rendered with our algorithm. An approximately 12.5% reduction of memory usage is gained compared to the standard OIT algorithm. More than 4 million fragments were used to generate this image (rendered at 14 frames per second).

Both equations use the following symbols:

C'_{acc} - accumulated color in the pixel
a_{acc} - accumulated opacity in the pixel
C_i - color of sample S_i
a_i - opacity of sample S_i.

The advantage of the latter form of the equations is that the alpha channel can saturate quickly, saving computation time by reducing the number of processed samples

3. Linked Lists and OIT Variants

The linked list OIT on the GPU is the first efficient implementation of OIT. It was first published in its current form by J. Hensley, AMD Research [8].

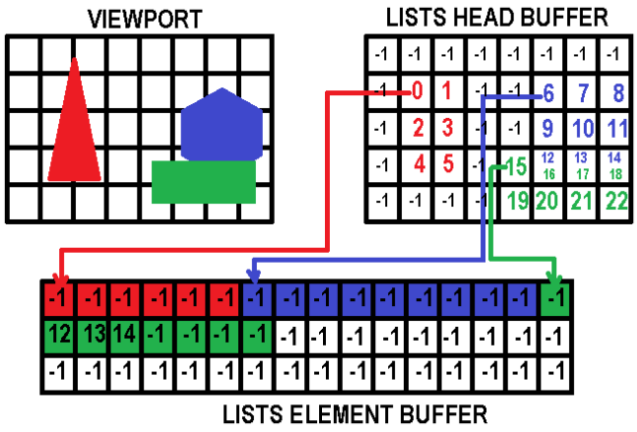


Fig. 2. Example Basic OIT with linked lists: the lists head buffer contains the heads of each pixel list, while the element buffer contains all the elements (i.e. samples) from all pixels

The linked list algorithm is a two phase algorithm that can be implemented as a single pass algorithm using indirect rendering. It is based on the presented theory and is in practice just a GPU implementation of the A-buffer [9]. As can be seen in Fig. 2, the algorithm requires two buffers: the lists head buffer, which is used to store the first element for each of the per pixels lists and the element buffer, which is used to store the rest of the elements for each list. Each element in the element buffer list points to another element in the buffer list or to -1, which is a convention for the end of the list.

During the first phase both buffers are completed with the depth, color and transparency of the scene fragments and in the second phase this information is extracted as a list from the buffers, it is sorted per pixel using the depth and then it is composited using the front to back or the back to front equations.

There are many variants of the base algorithm: for example the F-Buffer[10] which employs a multi-pass algorithm that requires a global sort on all the fragments. The A-Buffer and F-Buffer are similar algorithms with similar memory requirements, but the A-Buffer only needs per pixel sorting while the F-Buffer requires per framebuffer sorting which results in much better parallelism of the former.

Kbuffer[11], Stencil Routed Kbuffer [12] and Bucket Sort Depth Peeling[13] are all variants of depth peeling and all require multi-pass techniques. While the A-buffer variant can solve the blending equations in a single rendering pass, the depth peeling variants need at least L/M passes where L is the number of layers and M is the number of multiple render targets supported. They can be considered as advanced variants of a memory limited A-Buffer, but are not as efficient in memory usage.

. Tiled linked list OIT is a method in which the atomic counters required for the synchronization process are distributed in tiles, reducing the number of accesses per counter, thus distributing the synchronization effort and improving performance.

Another variant was introduced which allocates memory in pages and thus eases access to memory.

The stochastic perspective on transparency adds another dimension to the OIT field, for example adaptive transparency by M. Salvi et al[7] uses visibility function approximations in order to better determine sample importance, thus improving the memory usage.

4. Algorithm Overview

Our proposed algorithm is a two phase algorithm. In the first phase the lists head buffer and the lists element buffer are filled with information from the samples obtained through rasterization. In the second phase of the algorithm the samples are extracted from the buffers are then sorted by depth and finally are combined through the front to back equations. Using indirect rendering, the algorithm runs in a single pass. The algorithm keeps the closest known sample as the head of the per pixel list and then, for each of the upcoming samples it tests its contribution by considering a list of only the head, the next element in the list (if any) and the shaded sample.

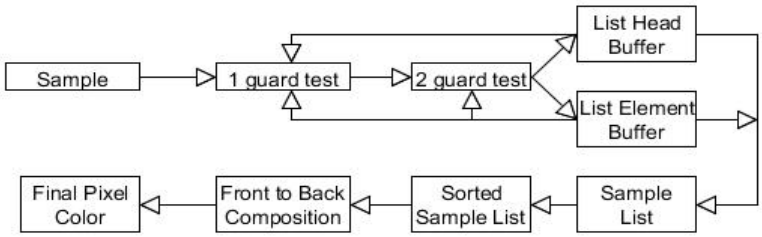


Fig. 3. The Sample Rendering Pipeline

As shown in Fig. 3, the pseudo code for the per fragment run algorithm is:

→ Create sample S_i from shaded fragment at xy screen coordinates.
→ If head is not empty
  → If the head H_{xy} is closer than S_i
    → If importance(S_i, H_{xy})<T
      → Exit
    → If there is sample S_k closer than S_i
      → If importance(S_i, H_{xy},S_k)< T
        → Exit
    → Save sample
  → Else
    → Introduce new head data and add old head as a new sample to the element buffer
    → Exit
→ Else
  → Introduce head data

The importance function is a comparison between the contribution of the fragment to the pixel and a threshold T, as described in Equation 1.

The guard algorithm is integrated with ease in the normal OIT method, the guarded OIT just adds the guard function to the fragment shader in the first phase of the OIT technique.

Even if the algorithm works well for general scenes – more on that in the results section – some particular yet common scenarios led to the concept of a fragment guard. For example scenes where complex materials are rendered where parts of the material are transparent and others are almost opaque can't be rendered efficiently with order independent transparency or sufficiently correct

with the opaque pipeline. Materials that are included in this category are mosaics, painted glasses or car windows.

5. Implementation

While the technique is clearly explained in the algorithm overview section, some aspects of order independent transparency need more clarification. Efficient implementation is rarely discussed with OIT, and there are a number of pitfalls such as correct synchronization, efficient computation, or efficient draw setup.

Correct synchronization is needed because even if we are using atomic operations and are using coherent memory, two threads can possibly access the same area of memory in a very short interval which can create read-write conflicts. Thus, a spinlock is required.

A simple spinlock implementation on the GPU uses an additional state buffer, the contents of which will act as the value of the spinlock used to synchronize the GPU threads. For example if the value in the buffer is 0 then the spinlock is free, if the value is 1 then the spinlock is taken.

Both the acquire and release spinlock methods can be implemented by using atomic operations. For example in GLSL atomicCompSwap might be used for conditional comparisons in the acquire method and the atomicSwap for simple writes in the release method.

Even though a single spinlock can successfully service a large number of threads with the help of the latency hiding mechanisms in modern GPUs, the usage of a single spinlock is not sufficient to prevent starvation, thus many need to be used for efficient running.

Efficient synchronization can be obtained through tile based OIT, in which many spinlocks are employed, each for a single tile in the frame buffer. Unfortunately atomic operations performance does vary a lot between the different GPUs targeted OIT methods and there is nothing that can be done about it. Furthermore many drivers will terminate a shader invocation if it lasts too long, therefore extra care must be taken while testing in order to correctly balance the synchronization effort.

Even if in general the visual results may not vary between OIT with spinlocks and OIT without them if the scene is not enormous or if the scene uses expensive materials which make the potential read-write race rare, spinlocks become a necessity when using the guard variant presented in this paper.

Efficient computation can be achieved through front to back sorting and compositing only when the sample is not occluded by an already saturated alpha channel. A naive scene object sort can offer a great setup for our algorithm because if the objects with the fastest alpha saturation potential are drawn at the beginning this will offer more chances for fragment culling. Since any renderer

will include such a step, it can't even be considered as part of the algorithm but more of an implementation observation. It is important to note that this algorithm is implementable only on Shader Model 5 hardware and that it requires unordered reads and writes from the GPU.

6. Results

The test GPU was a GTX460M. The guard threshold was set to 0.25% of the pixel value. As it can be observed from Fig. 4, the algorithm can offer substantial memory and some computational relief, especially to massive scenes with lots of fragments.

Scene name and View	Percent of fragments culled	Fps Without Guard	Fps With Guard	Fps Percent Gain
Crytek -1	7.28	7.12	7.08	-0.56
Crytek -2	15.81	6.64	6.68	+0.6
Crytek -3	26.39	7.7	8.41	+9.22
Dabrovic-1	5.26	20.17	20.74	+2.82
Dabrovic -2	9.55	13.02	12.53	-3.71
Dabrovic -3	14.97	12.2	12.88	+5.71

Fig. 4. Comparison of results between linked list OIT and our implementation.

Rendering of mosaic, like other complex materials in which some parts are opaque and others have varying opacity, are very hard to render efficiently. The choices for rendering such a material are to render it twice with different material and shader states, which can be very costly. The algorithm treats such cases culling more than 40% of the fragments, while needing just a single render pass. An example of mosaic rendering is shown in Fig. 5.

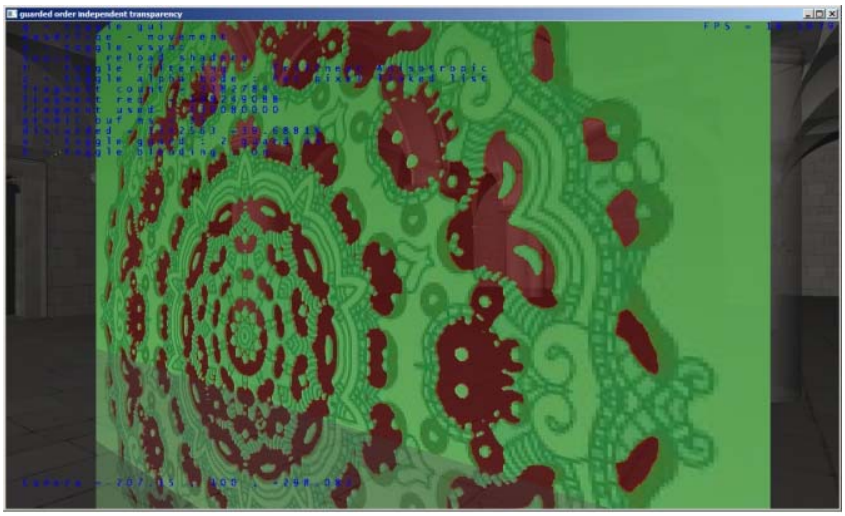
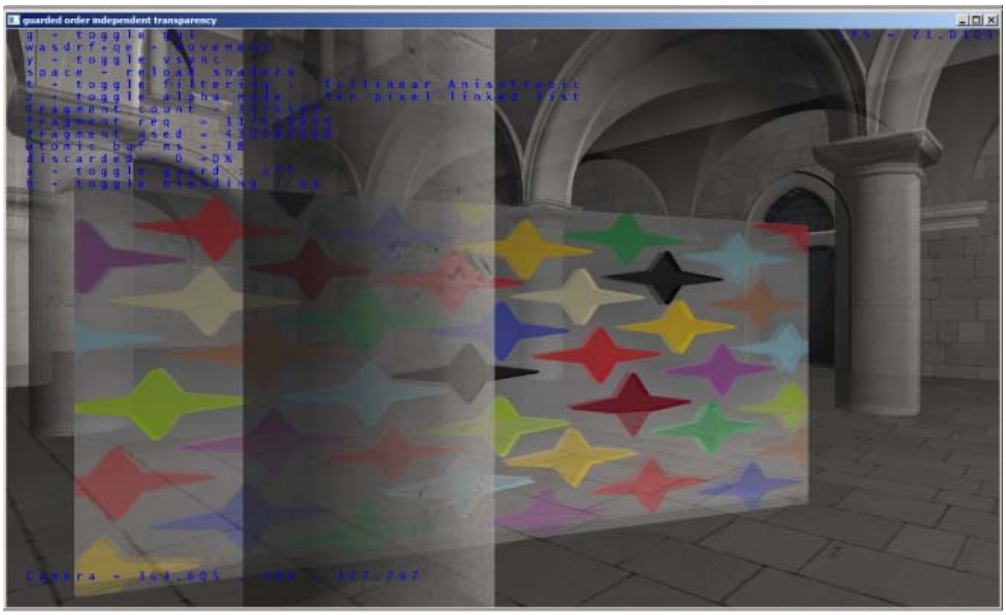


Fig. 5. Rendering mosaic like materials: partially close to opaque, partially transparent. This type of material is common for objects which include opaque and transparent components, such as cars. In this case the algorithm is significantly faster than other OIT methods while also working in a single rendering pass.

Due to the almost insignificant importance of poorly lit fragments in a transparent setup our algorithm offers increased performance in poorly lit scenes. Also, even by discarding a large number of fragments our method produces virtually the same image as the standard OIT algorithm. An example is offered in Fig. 6.



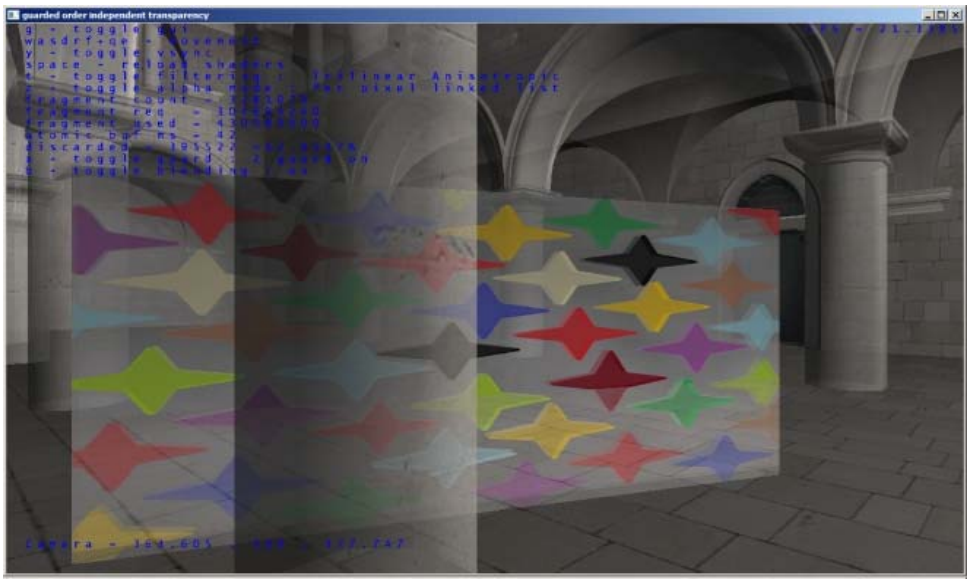


Fig.6. Comparison of results between linked list OIT and our implementation. A glass wall has stars on each of its sides, but some of the stars are not identical on each side in order to emphasise that opacity is obtained through the composition of the front face and back face of the stars on the glass. The difference between the images is minimal.

7. Discussion

Even though the majority of Shader Model 5 hardware offers better performance and notably better atomic operations performance than the GTX460M we used as testing hardware, the algorithm still performs at an adequate rate from a computational point of view.

From a memory point of view culling 7-25% of the total number of fragments in regular scenes is a lot since the testing scenes generate between 4 and 10 million of fragments.

The algorithm is particularly efficient for complex materials which combine almost opaque properties with a transparent look, such as painted glass, mosaics or dark car windows. In practice such materials are ubiquitous. Moreover the same property is valid for volumetric rendering, where the guard test can be used to even not ray march the volume, which can lead to excellent performance improvements. The performance of our fragment guard culling algorithm ranges from 12% to more than 40% for these particular materials. A particularly interesting property of the proposed algorithm is that it offers increased performance in poorly lit scenes. Furthermore, the algorithm should increase in performance with future hardware generations with faster atomic operation.

8. Conclusions

We have presented a memory efficient, technique independent, linked list order independent transparency algorithm. By using the 1-guard and the 2-guard with parameters that can be easily tweaked, the presented technique is able to cull between 7% and 25% of the total number of fragments of general scenes while practically outputting the same visual result.

The algorithm also handles well particular materials such as mosaic, painted glass and volumetric with increased efficiency, especially if used together with a naive scene object sort, found in almost any renderer. This eases the rendering process because fewer rendering passes are necessary and fewer programming states needs to be managed. We have proved that the presented algorithm is able to integrate with other OIT linked lists algorithms such as the tiled variant. Furthermore, integration with other related methods such as the F-Buffer is easy.

Because of the ever increasing difference between slower memory and faster processing units, memory optimizations and lossless color space compression algorithms [14][15] should always be the priority for list-based algorithms. Efficient implementation of OIT is particularly challenging because there are not many opportunities to decrease memory costs, especially when using lists, therefore the proposed algorithm adds a useful and novel technique to the OIT family.

Acknowledgement

Part of the research presented in this paper was supported by the Sectoral Operational Program of Human Resources Development, 2014-2015, of the Ministry of Labor, Family and Social Protection through the Financial Agreement POSDRU/159/1.5/S/134398 between University POLITEHNICA of Bucharest and AM POS DRU Romania.

REFERENCES

[1] *T. Porter and T. Duff*, "Compositing Digital Images" Computer Graphics Volume 18, Number 3, pp 253-259, July 1984

[2] *C. Everitt*, "Interactive order-independent transparency". Tech. rep., NVIDIA Corporation, 2001.

[3] *L. Bavoil, K. Myers*, "Dual Depth Peeling". Tech. rep. NVIDIA Corporation, 2008

[4] *J. Yang, J. Hensley, H. Grun, N. Thibieroz*, "Order Independent Transparency with Linked Lists", Computer Graphis Forum, Volume 29, issue 4, pages 1297-1304,June 2010

[5] *M. Maule, J.L.D Comba, R.Torchelsen, R. Bastos*, "Memory-Efficient Order-Independent Transparency with Dynamic Fragment Buffer" Graphics Patterns and Images SIBGRAPI 2012

[6] *E. Enderton, E. Sintorn, P. Shirley, D. Luebke*, "Stochastic Transparency", *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, February 2010

[7] *M. Salvi, J. Montgomery and A. Lefohn*. "Adaptive Order Independent Transparency: A Fast and Practical Approach to Rendering Transparent Geometry" , Game Developers Conference, March 2011.

[8] *J. Hensley*, "Order-Indepedent Transparency with DirectX11", Tech. rep., AMD Corporation, May 4. 2010.

[9] *L. Carpenter, L.* "The A-buffer, an antialiased hidden surface method". In Proceedings of SIGGRAPH, 103–108, 1984

[10] *W.R. Mark, K. Proudfoot*, "The F-Buffer: A Rasterization-Order FIFO Buffer for Multi-Pass Rendering" Preceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware, 2001.

[11] *L. Bavoil, S. Callahan, A. Lefohn, J. Comba, C. Silva*, "Multi-fragment effects on the GPU using the k-buffer" . In Symposium on Interactive 3D graphics and games,104, 2007

[12] *L. Bavoil, K. Myers*, "Stencil Routed K-Buffer", Tech. rep., NVIDIA Corporation, November 2007

[13] *F. Liu, M.C. Huang, X.H. Liu, E. W. Hua*, "Efficient Depth Peeling via bucket sort", Proceedings of the Conference of High Performance Graphics 2009, Pages 51-57, 2009.

[14] *A. Neves, A. Pinho*, "Lossless compression of color-qunatized images using block-based pallette reordering", Computer Graphics Science Volume 3211, pp 277-284, 2004

[15] *R. Radescu*, "Lossless compression tool for limited number of colors", U.P.B. Scientific Bulletin, Series C, Vol 71, Iss. 2, 2009