



AVSM:

## Adaptive Volumetric Shadow Maps



**Figure 1 – AVSM scene demonstrating 3 different emitters self-shadowing and shadowing each other.**

### Abstract:

Adaptive volumetric shadow maps (AVSM) is a new approach for real-time shadows that supports high-quality shadowing and self-shadowing from dynamic volumetric media for both forward and deferred rendering. Our sample focuses on smoke, but this technique could also be applied to hair and other similar, translucent media.

When rendering such effects, self-shadowing is a highly valued feature that gives the proper visual cues needed for defining the shape and structure of the media. In order to perform self-shadowing in volumetric media, it is required to be able to accumulate the partial occlusion value at any point in the volumetric object. This feature usually requires one of two methods: depth slicing or deep shadow maps. Depth slicing often suffers from noticeable slicing artifacts, has limited depth ranges, and/or is limited to one type of media. Deep shadow maps do not suffer from these problems, but are primarily used in off-line rendering systems as they have not been efficiently mapped to current real-time graphics hardware – often due to the unbounded nature of their data storage requirements.

In adaptive volumetric shadow mapping, the algorithm generates an adaptively sampled representation of the volumetric transmittance in a shadow-map-like data structure. Each texel of this shadow map stores a compact approximation to the transmittance curve along the corresponding light ray. The main innovation of AVSM is a new streaming, lossy compression algorithm that is capable of building a constant-storage, variable-error representation of visibility that can be used in later shadow lookups.

Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>



In this sample, a traditional DirectX implementation is provided. We also include a second implementation that uses a new feature of Intel graphics processors.

### **AVSM Algorithm overview**

The heart of the AVSM algorithm is the generation of a unique style of shadow map that encodes the fraction of visible light at specific distances from the light source. The fraction of visible light (transmittance) values are in the interval  $[0,1]$ . This algorithm consists of two major steps: a capture stage, and a compression stage.

#### **Capturing fragments**

The first step of the implementation is to render the media from the light source to capture a per-pixel linked list of all translucent fragments while drawing the media.

The per-pixel linked lists of light-attenuating segments are captured using DX11's support for atomic gather/scatter memory operations. The basic idea is that a first 2D R/W texture is allocated with integer values stored in each pixel. In this way, each pixel represents an index into the head of a linked list and is initialized to a 'null' value (for example, -1). These indexes encode an offset into a secondary and larger R/W buffer which stores all the linked list nodes. A pixel shader then allocates a node from the second buffer by atomically incrementing a globally shared counter whose value represents a pointer to a new node. It then inserts the new node at the head of the list by atomically swapping the previous head pointer with the new one. The content of the new node is updated and its next pointer is set to the previous head of the list.

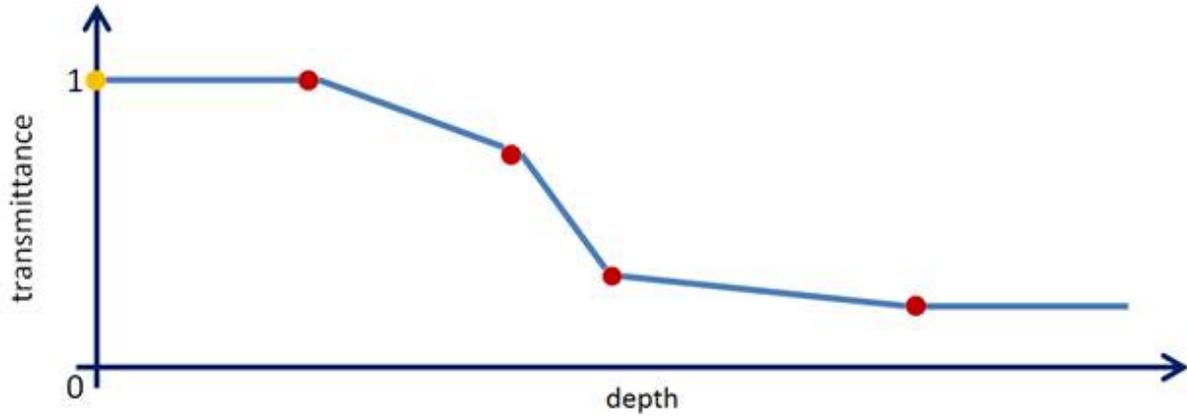
Since the capture phase must use a linked-list representation into a pre-allocated buffer; it is possible to run out of space while capturing the per-pixel fragments. This often shows itself as 'holes' or anomalies in the self-shadowing phase as not all the occluding fragments can be captured. This can obviously be fixed by making the buffer holding the capture phase larger; but this always presents the problem of limiting the effect or guessing the buffer size correctly while not wasting large amounts of memory. As we will see, however, an included version that utilizes a new feature called Pixel Shader Ordering does not suffer from this problem.

#### **Compression stage**

After the relevant fragments have been captured in our above mentioned data structures, we need to compute a compressed transmittance curve in a fixed amount of storage. Doing so saves not only memory and bandwidth, but also makes lookups very efficient. In general, the number of transparent fragments at a pixel will be much larger than the number of resultant compressed AVSM nodes, so we use a streaming compression algorithm that in a single rendering pass approximates the original curve.

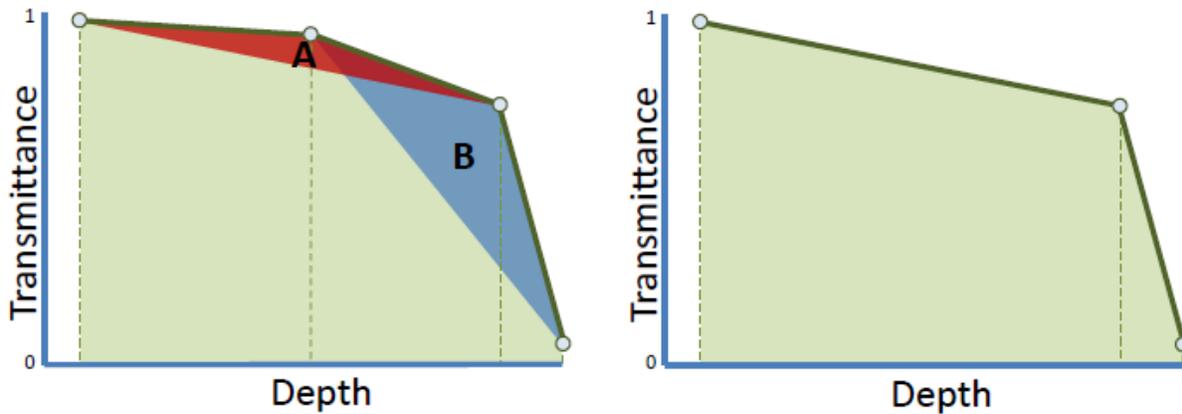
Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>



**Figure 2 – a curve representing light as it passes through volumetric media at a given pixel. Each red dot is a node that indicates light interacting with the occluding media.**

Each node of our piecewise transmittance curve maps to an ordered sequence of pairs that encode the node position (depth) along with an associated transmittance. This is stored as an array of pairs (depth + transmittance) using two single-precision floating point values. In this way, it is possible to query the data structure at a particular depth, and retrieve a transmittance value.



**Figure 3 – Compression of AVSM transmittance and depth data using an area-based reduction method**

During compression, AVSM computes the transmittance along a light ray by using a simple area-based curve simplification scheme. Figure 3 depicts compressing a 4-node curve into a 3-node curve. The algorithm simply removes the node that results in the smallest change in area under the curve, determined by computing the area of the triangles between the candidate node and its adjacent neighbors (triangles A and B). In this example, we see that node at A was removed because triangle formed with node A is smaller than the triangle formed with B. The number of nodes that one wishes to compress into is up to the developer – and our sample demonstrates 4 and 8 node versions.

Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>



Since this phase of the compression is lossy, ordering of the nodes matters. Inconsistent ordering can result in temporal artifacts; although they are mostly imperceptible in practice when using eight or more AVSM nodes. To avoid these problems in particularly complex media – sorting the segments via insertion sort can help reduce this problem.

### **Sampling from the AVSM data structure**

Once the map is generated, it acts very much like a standard shadow map. In fact, sampling from the AVSM structure can be seen as a generalization of a standard shadow map depth test. Instead of a simple binary depth test as done in traditional mapping, we evaluate the transmittance at the specified depth and get a value between [0, 1]. We look this information up using the above described compressed data structure.

Because of the irregular and non-linear nature of the AVSM data, we cannot rely on texture filtering hardware and must use a shader-based software filtering technique. For a given texel and depth, we perform a search over the entire domain of the curve for a given pixel to find the two nodes that bound the shadow receiver at the desired depth. We then simply interpolate between the two nodes to the desired depth to get our transmittance value.

### **Optimizing and taking advantage of new Intel features**

Provided in this implementation are both a DirectX version and a version that uses a new Intel graphics feature called Pixel Shader Ordering. The purely DirectX version is implemented as the algorithm is described above with two phases. The Intel-specific version that utilizes a new Pixel Shader Ordering feature allows serialized read-modify-write access to UAV's. This feature allows us to skip the first phase of the algorithm and completely avoid the need to create the linked-list fragment data structure. Instead, one can build the compressed node version directly from incoming fragments since Pixel Shader Ordering ensures no two shaders are working on the same pixel at the same time. This obviously reduces the amount of time, memory, and bandwidth required for the algorithm. It also means that it does not suffer from the memory limitations of the capture phase needed for the pure DirectX version. It simply inserts the fragments into the compressed AVSM curve using the streaming compression method for that pixel - so no fragments are accidentally overwritten or lost.

Further information about this new graphics feature will be available in another publication.

### **Sample particle system:**

The particle system used in our sample consists of billboards that evaluate to a simple spherical particle. It was only developed to the point that was required to show off a simple particle shadowing technique.

Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>



Much more interesting effects could be made by using a more complex and more physically based particle and volumetric media systems.

The particles themselves have a very simple spherical transmittance for the purposes of demonstrating this technology – but there is no reason one could not have a much more complex particles of varying size/shape/opacity.

### **Tessellation**

One optimization that improves performance and yields good visual results is to use shader tessellation with per-vertex lighting. To avoid more expensive per-pixel lighting calculations, we can tessellate the particles using the hull and domain shaders and then use faster per-vertex lighting evaluation. The results are nearly identical visually, and performance is improved. You can visualize the actual tessellation by selecting the wireframe particles checkbox in the sample.

### **Debug view**

A useful debug view is also provided to visualize the structure of the AVSM data. It acts like a 2D accumulation view into the AVSM depth and transmittance structure. As you move the slider forward and backwards, you control the 'depth' at which you are visualizing the volumetric shadowing from the lights point of view. Moving the slider further to the left moves the depth slice closer to the light. Moving the slider to the right moves the depth slice further away. The volumetric shadowing is cumulative, which means that as you expand the depth further and further back, it includes the shadowing of all the particles in front of it.

### **Future work:**

There are plans to explore several new areas for this sample. One area for exploration would be to use much more physically accurate particle systems that might employ different kinds of particles besides simple spherical particles. Media that consisted of numerous different types of media with different kinds of transmittance functions would also be very interesting. Using the data structure for other depth-based techniques could also yield new kinds of visual techniques.

### **References:**

#### **Adaptive Volumetric Shadow Maps**

<http://software.intel.com/en-us/articles/adaptive-volumetric-shadow-maps>

Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>



**Unordered Access Buffers:**

[http://msdn.microsoft.com/en-us/library/windows/desktop/ff476335%28v=vs.85%29.aspx#Unordered\\_Access](http://msdn.microsoft.com/en-us/library/windows/desktop/ff476335%28v=vs.85%29.aspx#Unordered_Access)

Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice

For more whitepapers and samples like these, visit: <http://software.intel.com/gamecode>