Eugene Kim

ABSTRACT

This paper introduces an algorithm to optimize the order that triangles are rendered in animated meshes. The objective of this algorithm is to reduce overdraw while maintaining good cache efficiency. Based on other algorithms designed for static meshes, the proposed algorithm clusters the space of viewpoints and generates a triangle in that cluster that has low overdraw and satisfactory cache efficiency.

INTRODUCTION

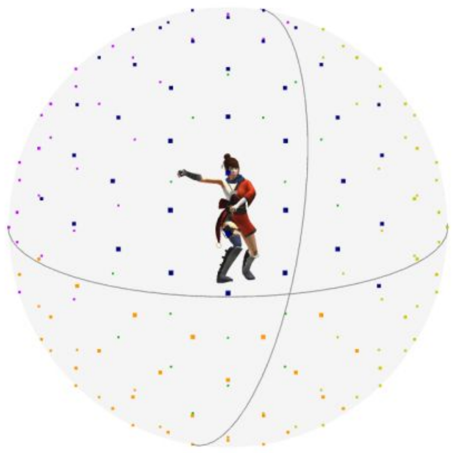
The largest bottlenecks with regards to rendering are either the vertex shader (vertex-bound scenes) or fragment shader (fill-bound scenes). Recomputation of a vertex can be avoided if the vertex was recently processed by an adjacent triangle using the same GPU, through its post-transform vertex-caching mechanism. This encourages triangles that share vertices to be near each other in the index buffer. ACMR, or average cache miss ratio, measures the ratio between number of vertices processed and number of triangles rendered. OVR, or overdraw ratio, refers to the ratio of the total number of fragments that pass a depth test to the number of visible fragments. ACMR and OVR are both directly proportional to rendering times for vertex-bound scenes and fill-bound scenes respectively.

GENERATING CACHE-EFFICIENT PATCHES

The algorithm initially follows a fast linear clustering approach to quickly get patches of triangles. The mesh is optimized to reduce ACMR by placing overlapping vertices near each other and the output index buffer was broken into contiguous triangle patches. A parameter, lambda, is used to regulate the ACMR by comparing the current patch's ACMR to the lambda, and adding a patch break whenever it drops below lambda.

GENERATING THE INDEX BUFFERS

A sphere that encloses the model is used to generate a set of 162 viewpoints which lie on the sphere's surface to represent the viewing locations. As the number of viewpoints increase, the more accurate the results at the expense of additional preprocessing time. After 162 viewpoints, the decrease in overdraw is outweighed by the processing required and therefore does not significantly improve results. With regards to significant global transitions, like a character running, the model's bounding sphere is calculated in each frame and the model is translated through them.



A view configuration consists of a viewpoint and the current frame. A node is created for each configuration so that each possible frame and viewpoint combination is fulfilled. For each configuration, an available index buffer with a low OVR is needed. A single grouping that contains every node cannot reduce overdraw, so k node clusters that can share index buffers are created. Therefore, there are k index buffers. The rendering algorithm selects the appropriate one at runtime based on the viewpoint and frame.

The algorithm is initialized by selected k initial viewpoints and creating an index buffer for each of them by sorting the mesh patches in front-to-back order (increasing distance between the patch center and the viewpoint). These k buffers represent the k clusters. The algorithm then alternates between two steps that assigns nodes to clusters and computes a new index buffer for each cluster.

Step 1: The scene is rendered using each of the k index buffers for each node at each node's frame and viewpoint, and uses hardware occlusion queries to read the overdraw results. Each node is assigned to the cluster whose index buffer gives the lowest overdraw.

Step 2: A new order that roughly sorts the patches front-to-back is created for each cluster, which allows you to compute a new triangle order with reduced average overdraw for every node assigned to the cluster. It is not as straightforward to sort the patches when multiple nodes are considered, so an integrated distance for each patch over every node n in the cluster c is computed, which results in a value that allows for sorting many nodes.

$$d(p) = \sum_{n \in c} \mathrm{dist}(p, v_n, f_n)$$

vn and fn are the viewpoint and frame associated with that specific node, and the dist() function returns the distance between vn and the patch p's centroid at frame fn. A mostly front-to-back order is created by sorting the patches in increasing order by d(p).

Pseudo code:

Algorithm 1

```
procedure CLUSTERNODES
Preprocessing:
    $F \leftarrow$ load animation frames
    $V \leftarrow$ generate representative viewpoints
    $N \leftarrow$ generate nodes from $F$ and $V$
    $P \leftarrow$ generate vertex-cache optimized patches
    for $f \in F$ do
        translateToOrigin(f)
Bootstrapping:
    $C \leftarrow$ generate initial viewpoints for each cluster $c \in C$
    for $p \in P$ do
        averagePatch(p) compute average patch vertex positions over all frames
    for $c \in C$ do
        $IB_c \leftarrow$ computeBuffer($c$) sort average patches to generate index buffer
Node assignment:
    for $n \in N$ do
        assignCluster($n$) assign $n$ to $IB_c$ that gives smallest overdraw
    if no cluster assignment has changed then
        end
Index buffer computation:
    for $c \in C$ do
        for $p \in P$ do
            $d_p = \sum_{n \in c} dist(p, v_n, f_n)$
        $computeIB(c)$ update $IB_c$ using the the patch distances computed above
    goto Node assignment
```

Simply using the central viewpoint or central frame of the cluster, or both, to generate the order for every node would reduce processing time significantly, but it is not always suitable, as in the case with a rotating model component. Different viewpoints can lead to different orders among frames due to the varying orientations, so this algorithm considers every frame and viewpoint equally.

RUNTIME SELECTION

The application selects one of the k index buffers during rendering through a lookup table indexed by frame and viewpoint. This lookup time is negligible compared to the rest of the algorithm.

RESULTS

Increasing the number of patches by increasing lambda and having a larger number of triangles both increased preprocessing times. However, the paper did not attempt to heavily optimize preprocessing for speed. Rendering time ended up being directly proportional to ACMR for vertex-bound scenes and OVR for pixel-bound scenes, which can both be controlled by setting the lambda parameter. A lambda value between 0.75 and 0.95 provided a good balance between ACMR and OVR for the animated scenes. Lambda value of 0.85 and 3 were used for other testing. Increasing the number of clusters reduced overdraw in exchange for memory to store the additional index buffers. Using more than five clusters only yielded modest reduction, and therefore five was generally used.

With a lambda value of 3, the OVR values were consistently lower than other algorithms, but the ACMR values went very high. A lambda value of 0.85 yielded a less drastic decrease, but the ACMR values were at a more reasonable point.

Paper:
http://jcgt.org/published/0006/03/03/paper.pdf