AMD

The future is fusion

Tootle Version 2.2
A Mesh Optimization Library

May 27th 2009

Budirijanto Purnomo

Based on the research by
Pedro V. Sander, Joshua Barczak and Diego Nehab

Introduction

Tootle is a mesh optimization library developed by AMD that provides these optimizations:

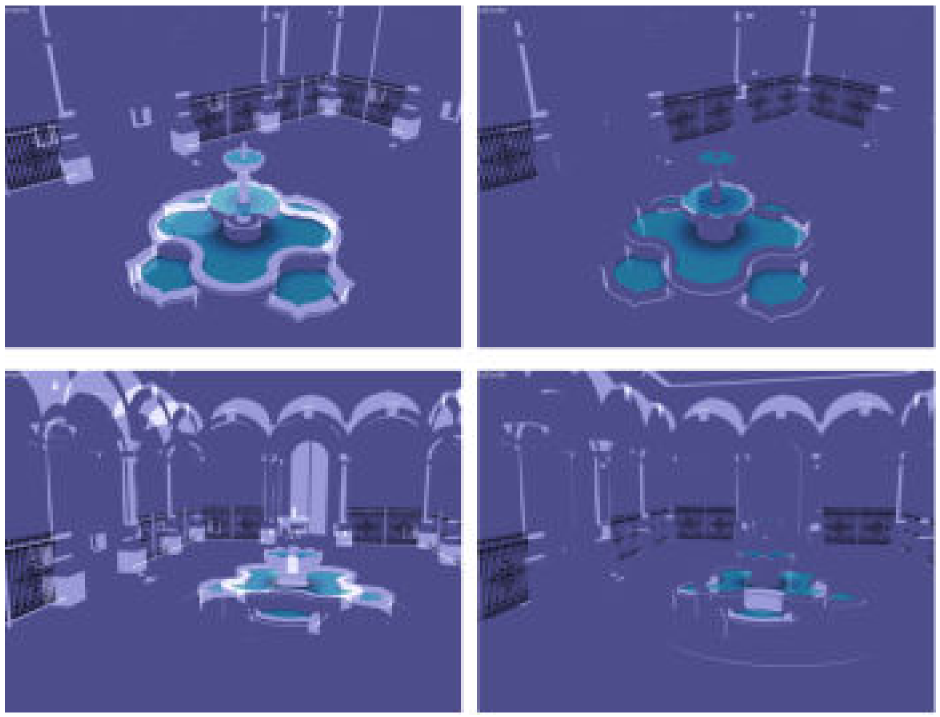
1. **Vertex cache optimization**: Triangles are re-ordered to optimize for the post-transform vertex cache in modern GPUs. This will yield significant performance improvements in vertex-tranform limited scenes.
2. **Overdraw optimization**: To reduce the pixel cost of rendering a mesh, the Tootle library further re-orders the triangles in the mesh to reduce pixel overdraw. Significant reductions in pixel overdraw (2x or higher) can be achieved. This can yield significant performance improvements in pixel-limited scenes, and incurs no penalty in vertex-limited scenarios.
3. **Vertex prefetch cache optimization**: Triangle indices are re-indexed in the order of their occurrence in the triangle list. The vertex buffer is re-ordered to match these new indices. Thus, vertices are accessed close to each other in memory. This optimization exploits the input vertex cache because vertices are typically fetched in a cacheline (that may contains more than one vertex data).

The Tootle library prior to version 2.0 (version 1.2 and lower) was based on the paper published and presented at I3D 2006 [1]. For version 2.x, Tootle includes the algorithm published and presented at SIGGRAPH 2007 [2] and also includes several new pieces of functionality. The new interface in version 2.x allows application developers to control a specific code path of the mesh optimization algorithm. For example, it is now possible to choose among the four different code paths for the vertex cache optimization (which is not possible in v1.2). They are: (1) using Direct3D face optimizer, (2) using triangle strips like list optimizer (good for small vertex cache size), (3) using the tipsy optimizer from SIGGRAPH 2007 (this is the default setting), or (4) using the auto setting similar to v1.2 where it will choose between the triangle strips like list or the tipsy optimizer depending on the vertex cache size.

In general, you can expect an average of 1000x run-time improvements of running Tootle when you use the function calls from version 2.x. However, the resulting mesh might not be as good as the one generated from the previous version of the library especially for overdraw optimization. As a recommendation, we suggest always using the new function calls from version 2.x initially to optimize your mesh. Then, you can perform the old library function calls for further improvements if needed.

While we made many changes to the interface from v1.2 such as adding more entry points and extra parameters to the old entries, programs developed using Tootle v1.2 will still compile with v2.x header and libraries. This is ensured by having all the extra parameters added to the old entries with their default values. However, there is *one important change* that application developers have to make to their applications. If you allocated a buffer of nTriangles (the total number of triangles in the mesh) for the cluster array IDs (the array used by TootleClusterMesh() and TootleOptimizeOverdraw()) in your application, the buffer has to be allocated of size nTriangles+1 for v2.x Tootle library. This is a necessary change because of the different buffer formats used by the cluster array IDs in v1.2 and v2.x. The good news is it is possible to mix-match the function calls from v1.2 and v2.x with these different buffer formats. The library will take care of detecting and converting from one format to another format whenever necessary (this will be transparent to the application developers).

For details on the algorithm, and more detailed results, we encourage you to check out our papers (too-paper.pdf and too2-paper.pdf) and our talk slides (too-talk.pdf and too2-talk.pdf).



Visualizing overdraw in a typical scene. Bright areas have high overdraw. Left: Vertex Cache Optimization alone. Right: Using Tootle together with vertex cache optimization. Even with very simple pixel shaders, performance gains of three to eight percent were observed.

Changes since Tootle v1.2

- There is *one necessary change* for your old application to work with Tootle v2.x.
  If you allocated the cluster array IDs of size nTriangles (the total number of triangle in the mesh) before, it has to be changed to nTriangles+1 elements. Otherwise, the application will crash.
- *New entry* functions are added:
  1. TootleFastOptimizeVCacheAndClusterMesh(): perform fast optimization and clustering based on the algorithm from SIGGRAPH 2007,
  2. TootleOptimize(): a utility function to perform the entire mesh optimization based on the algorithm from I3D 2006 except that the overdraw optimization is defaulted to the algorithm from SIGGRAPH 2007,
  3. TootleFastOptimize(): a utility function to perform the entire mesh optimization based on the algorithm from SIGGRAPH 2007, and
  4. TootleOptimizeVertexMemory(): perform vertex prefetch cache optimization by re-ordering the vertex buffer and re-indexing the triangle indices.
- *Extra parameters* are added to the old entries from v1.2 to control the specific code path of the algorithm. A new parameter (an enumeration) to choose among four different possible code paths of vertex cache optimization algorithm (AUTO, TIPSY, LSTRIPS and DIRECT3D) is

added to the function TootleOptimizeVertexCache(). For the function TootleOptimizeOverdraw(), you can choose among FAST_APPROXIMATION (the algorithm from SIGGRAPH, this is the default setting), DIRECT3D (rendering from a set of viewpoints using Direct3D calls), RAYTRACE (rendering from a set of viewpoints using software rendering path), or AUTO (rendering from a set of viewpoints using either Direct3D or software rendering path depending on the number of clusters generated, this is the old setting from v1.2 library).
- An *updated* sample application. There are five cases in the application to show how to use the various function calls in the library. These cases can be controlled from the command line interface.
- There is support for several platforms: Windows XP 32 bit, Windows Vista 64 bit and Linux. There is also a library for pure software Tootle (without Direct3D dependency) under Windows.

What's in the Box

This package includes:

- A library that can be linked to your mesh processing pipeline and provides a very simple interface to optimize the triangle order of your mesh. We provide libraries for multiple platforms (Windows 32 bit, Windows Vista 64 bit and Linux).
    1. Windows:
        - DLL version: Tootle.dll and TootleDLL.lib
        - Static library versions: TootleStatic_MT.lib and TootleStatic_MTDLL.lib
        - Software only versions (without Direct3D dependency): the filenames are appended with SoftwareOnly
        - Debug version: the filenames are appended with _d
        - 64 bits version: the filenames are appended with 64
        - Visual Studio 2008 version: the filenames are appended with 2k8
    2. Linux:
        - Static library versions: libTootle.a (optimized), libTootle_d.a (debug)
- A sample application that reads a *single material* triangle mesh file .obj and exposes the functionality of the library to optimize the mesh using a command line interface. There are two visual studio projects for this sample application. One links to the library with Direct3D dependency, another links to the Tootle library without Direct3D dependency. A Linux build (Tootle and Tootle_d) with a makefile is also included.
- A sample application that loads a COLLADA document and applies tootle overdraw optimization and vertex cache optimization to each mesh in the document using a command line interface.
- Various sample meshes.
- Various pieces of documentation.

The Tootle Library

The Tootle library provides entry-points for the following:

1. Vertex cache optimization.
2. Mesh clustering.
3. Overdraw optimization of clustered meshes.
4. Measuring vertex cache efficiency and overdraw.
5. Vertex prefetch cache optimization.

For more information about using Tootle, we recommend examining the provided sample application. Detailed HTML documentation for each of the Tootle library functions is also provided.

Optimizing Face Order

In order to optimize a mesh, an application must first group its faces into clusters. The TootleClusterMesh() function may be used for this purpose, or the application may provide its own clustering.

Once the mesh has been clustered, the application should perform vertex cache optimization within the mesh clusters. This can be done by iterating over the clustered index buffer returned by TootleClusterMesh() and calling TootleOptimizeVCache() on each subset. Tootle provides a helper function, TootleVCacheClusters() to simplify this process.

New to v2.x is the addition of new entries TootleFastOptimizeVCache() and TootleFastOptimizeVCacheAndClusterMesh(). The former performs vertex cache optimization based on the tipsy algorithm from SIGGRAPH 2007. It is also possible to pass a parameter TOOTLE_VCACHE_TIPSY to the old function TootleOptimizeVCache() to perform the same algorithm. The latter performs the vertex cache optimization and mesh clustering in a single step based on the algorithm from SIGGRAPH 2007. There is not a separate entry to cluster the mesh based on the algorithm from SIGGRAPH 2007 because the clustering algorithm works together with the vertex optimization algorithm to partition the mesh (see the SIGGRAPH paper [2] for more details).

After vertex cache optimization, the function TootleOptimizeOverdraw() is used to sort the clusters in a way that minimizes overdraw over a set of viewpoints. By default, Tootle will use a set of viewpoints centered on the unit sphere around the object, but a user-defined set of viewpoints may be used instead. The index buffer returned by TootleOptimizeOverdraw() can then be used directly to render the re-ordered mesh.

In v2.x, there are two new utility functions added to the library. They are used to perform the entire mesh optimization as described above. These are the TootleOptimize() and TootleFastOptimize() functions. The former is used to call the mesh optimization algorithm from v1.2 functions (except the overdraw optimization is defaulted to the fast approximation algorithm from SIGGRAPH 2007) and the latter is used to call the mesh optimization from v2.x functions.

Optimizing Material Order

If a mesh has multiple materials that must be partitioned into several distinct draw calls, then the above procedure can be performed separately for each material. After optimizing within each material group, it is often beneficial to call TootleOptimizeOverdraw() a second time, this time using the material IDs of the mesh faces as cluster IDs. The cluster re-mapping that is returned by TootleOptimizeOverdraw() can then be used to re-sort the material groups in a way that minimizes the overdraw between them. This technique is not implemented in the sample application, but a code example is provided in the file MaterialSort.cpp.

Usage Examples

The variables used in the examples below are defined as follows:

pVB	is a pointer to the vertex buffer of the input mesh
pnIB	is a pointer to the index buffer of the input mesh
nFaces	is the total number of triangles of the input mesh
nVertices	is the total number of vertices of the input mesh
nVBStride	is the distance between successive vertices in the vertex buffer in bytes
pnIBOut	is a pointer to the updated index buffer (the output)
nCacheSize	is the hardware vertex cache size in vertices

The following examples show the typical usage of Tootle in an application.

1. Using Tootle with the vertex optimization algorithm only:

```
TootleOptimizeVCache( pnIB, nFaces, nVertices, nCacheSize, pnIBOut, NULL,
                      TOOTLE_VCACHE_AUTO );
```

   The code above will use the AUTO code path for the vertex optimization algorithm which will be either using TIPSY (SIGGRAPH 2007 version) or LSTRIPS (triangle strips like list) algorithm depending on the hardware cache size.
   To force the library to use a specific code path of the vertex optimization algorithm, you can pass either TOOTLE_VCACHE_DIRECT3D, TOOTLE_VCACHE_LSTRIPS or TOOTLE_VCACHE_TIPSY for the last argument of the function.

2. Using Tootle with the algorithm from I3D 2006 (version 1.2 and lower):

```
TootleOptimize( pVB, pnIB, nVertices, nFaces, nVBStride, nCacheSize,
                NULL, 0,                  // use the default viewpoints
                TOOTLE_CW,                // the mesh has clockwise front face winding
                pnIBOut, NULL,
                TOOTLE_VCACHE_AUTO, // use the AUTO path for vertex optimization
                TOOTLE_OVERDRAW_AUTO );
```

The last argument to the function specifies the code path for overdraw optimization; the AUTO path will be either using DIRECT3D (hardware rendering using occlusion query) or RAYTRACE (software rendering) depending on the number of clusters generated. To force the library using a specific code path of the algorithm, you can pass TOOTLE_OVERDRAW_DIRECT3D, TOOTLE_OVERDRAW_RAYTRACE or TOOTLE_OVERDRAW_FAST (SIGGRAPH 2007 version). The last two arguments to the function are optional. If omitted, they will be defaulted to TOOTLE_VCACHE_AUTO and TOOTLE_OVERDRAW_FAST.

3. Using Tootle with the algorithm from SIGGRAPH 2007:

```
TootleFastOptimize( pVB, pnIB, nVertices, nFaces, nVBStride, nCacheSize,
                    TOOTLE_CW, pnIBOut, NULL, NULL, TOOTLE_DEFAULT_ALPHA
                    );
```

The last argument to the function is optional. It controls the number of clusters generated by the algorithm.

4. Using Tootle by mix-matching the algorithm from I3D 2006 and SIGGRAPH 2007:
   i. Allocate space for the cluster array IDs

```
unsigned int* faceClusters = new unsigned int[ nFaces + 1 ];
```

   ii. Cluster the mesh using the algorithm from I3D 2006

```
TootleClusterMesh( pVB, pnIB, nVertices, nFaces, nVBStride,
                   0,        // let the algorithm determines the target number of clusters
                   pnIB,     // overwrite the input index buffer
                   faceClusters, NULL );
```

   iii. Perform vertex optimization for each cluster

```
TootleVCacheClusters( pnIB, nFaces, nVertices, nCacheSize, faceClusters,
                      pnIB,   // overwrite the input index buffer
                      NULL, TOOTLE_VCACHE_AUTO );
```

   iv. Perform overdraw optimization using the algorithm from SIGGRAPH 2007

```
TootleOptimizeOverdraw( pVB, pnIB, nVertices, nFaces, nVBStride,
                        NULL, 0,          // use the default viewpoints
                        TOOTLE_CW, faceClusters, pnIBOut, NULL,
                        TOOTLE_OVERDRAW_FAST );
```

Step (ii) and (iii) can be replaced by TootleFastOptimizeVCacheAndClusterMesh() to use the clustering algorithm from SIGGRAPH 2007.

Notes on Overdraw Optimization Speed

Tootle contains three code paths for optimizing meshes for overdraw. One of these is a GPU-based path which uses occlusion queries to measure overdraw between pairs of mesh clusters. The second path uses a software rendering algorithm. The software rendering algorithm can be quite slow, but it is still significantly faster than the GPU path when the cluster count is large (we have observed the difference to be up to a factor of 10, in some cases). If the number of clusters exceeds a hardcoded threshold (currently 225), Tootle will use the software rendering path.
The third path uses a fast approximation algorithm from SIGGRAPH 2007. This path is significantly faster than either the GPU or the software rendering path. We have observed an average of 1000x run-time improvements. However, the resulting mesh might not be as good as using the previous two paths. As a recommendation, we suggest always using the fast approximation algorithm path to optimize overdraw initially. If you require better results, then run either the GPU or the software rendering path.

In some cases, a call to `TootleClusterMesh()` can generate a very large number of clusters. This is most common when there is little or no sharing of vertices between triangles. Whenever possible, the vertices in the mesh should be 'welded' prior to calling tootle. This will reduce the number of clusters that are generated, and will also result in better vertex processing performance. If vertex welding is not an option, and tootling within materials is too slow, it might be better to simply re-order the materials using tootle, without optimizing face order within materials (see *Optimizing Material Order*). Unless the material count is very high, material sorting will always use the hardware path.

Tootle Sample Application

The sample application (`tootle.exe`) is a simple example project that uses the library. It only reads a *single material* triangle mesh file (`.obj`) that has absolute vertex indexing. If you plan to use this Tootle in production, we recommend using the library directly in your pipeline after your materials have been split and vertex/index buffers have been constructed.

Syntax:
```
tootle inmesh.obj [-a <num: 1-5>] [-c <num>] [-f] [-m] [-o <num: 1-4>] [-p] [-s <num>] [-v <file>] >
        outmesh.obj
```

- `inmesh.obj`
  The obj file you wish to optimize.
- `-a <num: 1-5>`
  Optional parameter to choose among the five different cases for Tootle.
  They are:
  1. perform vertex cache optimization only,
  2. perform the entire mesh optimization using three separate v1.2 function calls,
  3. perform the entire mesh optimization using a single utility function for v1.2 function calls,
  4. perform the entire mesh optimization using two separate v2.x function calls, and

5. perform the entire mesh optimization using a single utility function for v2.x function calls.

  If omitted, the third choice will be performed.

- -c <num>
  Optional parameter specifying the number of clusters.
  In general, users will want to let Tootle cluster automatically, but manually setting a high cluster count can sometimes provide greater overdraw reductions, at the expense of poor vertex cache performance. Note that the cluster count is only a hint. If the mesh is highly disconnected, then the algorithm may need to generate a larger number of clusters. This parameter will only be used for the v1.2 functions.
- -f
  If supplied, front faces are assumed to have counter-clockwise winding order. If omitted, then front faces have clockwise winding.
- -m
  If omitted, the measuring overdraw function will be performed.
  If supplied, the measuring overdraw function will be skipped (useful for the software only build since this function may be slow using the software path).
- -o <num: 1-4>
  This optional parameter allows users to choose the four different code paths for the vertex optimization algorithm: (1) AUTO, (2) DIRECT3D, (3) LSTRIPS or (4) TIPSY.
  If omitted, AUTO (either LSTRIPS or TIPSY based on the vertex cache size) will be used.
- -p
  If omitted, the new vertex prefetch cache optimization will be performed on the output mesh.
  If supplied, the optimization will be disabled.
- -v <file>
  Optional parameter specifying the viewpoint file. This should be an ascii file containing the number of viewpoints in the first line, followed by the 3D coordinates of the viewpoints, one per line.
  The viewpoints in the file will be used in place of the default viewpoint set. Viewpoints must be points on or in the unit sphere. During overdraw measurement, the mesh will be scaled and translated to fit inside this unit sphere, and will be rendered using orthographic projections from each viewpoint, looking at the origin of the model.
  A user may want to use custom viewpoints to optimize a mesh if there are restrictions on the angles from which the object can be viewed. For example, if an object is always viewed from above, and never from below, then it is advantageous to place all of the sample viewpoints on the upper hemisphere.
- -s <num>
  The vertex cache size in vertices. If omitted, the default size 16 will be used.

Important Notes

- The libraries were compiled in Microsoft Visual Studio 2005 and 2008 for Windows XP 32 bits and Windows Vista 64 bits.

They were tested using the DirectX SDK from March 2009. We also provide a Linux version (compiled with GNU g++ under Ubuntu Linux) which is based on a pure software library (no Direct3D dependency). This software version is also provided for the Windows platform.
We have not tested the code on any other platform. Builds of the library are provided for Visual Studio 2005 and 2008 in both debug and release configurations. The main difference between these is the version of the CRT that they use. In addition, a DLL is also provided.
For the Linux builds of the library, we provide static libraries compiled with debug and optimized flag.

- For the Direct3D version of the library, Tootle's implementation uses Direct3D to repeatedly render the mesh in order to measure overdraw. Any applications which make use of the tootle library must also link against both d3d9.lib and d3dx9.lib (unless the DLL is used). Because Tootle uses Direct3D to repeatedly render the mesh, the optimization will be much faster if the D3D release runtime is used. The machine on which Tootle is run must have a GPU with occlusion query support (virtually all GPUs on the market today have this support), and for best results, the GPU should be fast.
- For the Linux library, you can build the sample application in Linux by typing `make` in the `samples` directory.
- Numerous changes, bug fixes, and improvements have been added to Tootle since the publications in SIGGRAPH 2007 and I3D 2006. For this reason, the results produced by this release of the tool may not match exactly with those reported in the original paper.
- As mentioned above, the default set of viewpoints assumes that the mesh can be viewed from all directions. A user-defined set of viewpoints can be used if the mesh can only be viewed from one side (e.g., if it is blocked by wall from the other side). However, we do not expect the additional gains from this to be significant except on very special scenarios.
- This tool does not support animated meshes. If you plan to use this on an animated mesh, you can consider applying the algorithm on a static pose that is indicative with respect to what parts of the mesh are likely to occlude other parts. For example, a model of a certain vendor-specific comic heroine should probably be Tootled with her arms at her sides, rather than having them outstretched. Using Tootle with the algorithm from SIGGRAPH 2007 path, you can potentially run the optimization after every pose changes. For moderate size models (ten of thousands of triangles), the optimization will complete in less than a second (typically in the order of hundreds milliseconds).
- Feel free to email us at gputools.support@amd.com if you have any questions, comments, or, especially bug reports.
- In case you are wondering, the name *Tootle* is inspired by the pronunciation of *T. O. O. Tool*.

References

[1] Diego Nehab, Joshua Barczak, Pedro V. Sander. *Triangle Order Optimization for Efficient Graphics Hardware Computation Culling*, 2006 ACM Symposium on 3D Graphics and Games, Redwood City, CA.

[2] Pedro V. Sander, Diego Nehab, and Joshua Barczak. *Fast Triangle Reordering for Vertex Locality and Reduced Overdraw,* ACM Transactions on Graphics (Proc. SIGGRAPH). 26(3) August 2007.