Subdivision Surfaces using Map-Reduce and the GPU

July 10, 2012, Posted by Paul Doyle

In this white paper, we examine how Catmull-Clark subdivision surfaces can be computed using Fabric Engine's map-reduce framework. We examine the performance of the resulting system, and also present results obtained by leveraging the GPU.

Catmull-Clark subdivision surfaces

In computer graphics, smooth surfaces are generally represented by bicubic surfaces, meaning that each of the two dimensions of the surface is represented by a cubic (third degree) polynomial. There are several ways to represent these surfaces, which primarily differ in how they are authored and manipulated by the user. NURBS (Non-Uniform Rational B-Splines), Bézier surfaces and Catmull-Clark surfaces are among the most common, the latter being particularly easy to work with, as they are essentially controlled by a low-resolution “hull” that is akin to a faceted version of the object the user is modeling. Catmull-Clark surfaces are computed by recursively applying the algorithm to the hull, which is progressively smoothened with each recursion level (for the details of the algorithm, see E. Catmull and J. Clark: Recursively generated B-spline surfaces on arbitrary topological meshes, Computer-Aided Design 10(6):350-355 (November 1978)).

For high-quality rendering purposes, this recursion is to be performed until the surface appears completely smooth; however, during authoring, the recursion is typically limited to very few levels in order for the application keep the total number of polygons to draw in check. More importantly though, as the control hull changes, either through user interaction or animation data, the surface needs to be recomputed, and a low limit on the number of recursions is crucial to keeping acceptable performance. Here is a polygonal mesh, which is used as the hull of a subdivision surface:

And here is the resulting surface after two levels of recursion of the Catmull-Clark algorithm:
Here it is after 5 levels:

To modify the final, subdivided shape, the user actually only needs to manipulate the (far fewer) vertices of the hull (unsubdivided) mesh, which is typically drawn in wireframe around the object. Here is how this looks in Blender, a popular open source 3D content creation package:
Map-Reduce in Fabric Engine

Before we get to Fabric Engine’s implementation of subdivision surfaces, a brief overview of Fabric Engine’s map-reduce model is in order. In general, the map-reduce approach to computation asks that a problem be defined in a map operation and a reduce operation. The map operation takes an array of input values and for each value, produce an intermediate result – these map operations being parallelized. Then, the reduce operation gathers the results of all map operations and produces a single output value which is the answer to the problem.

Fabric Engine’s Map-Reduce expands on this concept and is based on Producers and Reduce operations. Producers produce values or arrays of values when requested to do so; they can either produce constant values, or call a function (supplied when the producer is created) to produce their values. For example, a constant Producer might always return the number 5, or the string “hello”. A function Producer might have been created with a function that returns the time of day, and so will return that.

Maps are another kind of Producer that take a value of one type as input and produce a value of a potentially different type as output (these types potentially being Producers themselves). For instance, a Map might take a string as input and return the string’s length as output.

Finally, Transforms are a special case of Maps where the output type is the same as the input type. A Transform might, for instance, return the square root of the number it is supplied.

Fabric Engine Producers can produce arrays, and those that do are able to return partial results; that is to say, rather than asking a producer to produce its whole output array, we can limit it to producing only elements 15 to 511 for example; this is of obvious utility for distributing computation across multiple cores.

Fabric Engine’s Reduce objects are specialized Producers that will, when asked for their results, ask the Producers it received at construction time to produce their values; Reducers then use the reduce function they were constructed with to accumulate the result; certain tweaks to the canonical MapReduce model allow Fabric Engine to not be required to hold the results of all Map operations before it starts the Reduce operation.

Subdivision surfaces as a series of Map-Reduce problems

Our subdivision surfaces implementation uses our Map-Reduce framework at 3 key stages of computation. In the first step, we find all the different topological configurations of polygons at a given subdivision level (such as, for instance, how many neighbors a polygon has). In this step, the Map consists in examining a polygon and returning its topology type; the Reduce operation then eliminates duplicate types and outputs a list of all topologies on the mesh without duplicates.

The second stage involves building a table for each of the topology types identified above; it produces a dictionary where the key is the topology type and the subdivision level, and the value contains information about that topology and its vertices: for example, how many vertices are generated, which ones are on edges, which ones are on corners, and what are the coordinates of each of vertices expressed as a linear combination of neighboring hull points. This is encapsulated in a Map operation, and because the output is the same size as the input, there is no Reduce operation to speak of.

The third stage, which is run every time the hull changes, is to compute the vertices of the subdivision surface proper, by computing the linear combinations calculated at the second stage. Again, there is no Reduce operation at this stage.

Results: Subdivision surfaces performance in Fabric Engine

For comparison purposes, we created geometries using the same number of triangles in the base mesh as our test asset in each of 2 of the most popular professional 3D content creation tools (Maya and Softimage). A 3DSMax license was not available when this test was conducted. We then measured the performance of the application as the hull (unsubdivided) mesh was animated; we then observed the resulting framerate and compared it to our face asset running in Fabric Engine. These tests were conducted on a 2011 MacBook Pro (4 cores) on windows 32 bit and the polygons were diffuse, smooth-shaded and untextured. Here are the results:
Subdivision level 1 (50k triangles):
- Softimage: 10 fps
- Maya SmoothProxy: 28 fps
- Fabric Engine on CPU: 60 fps
Subdivision level 2 (200k triangles):
- Softimage: 5 fps
- Maya SmoothProxy: 9 fps
- Fabric Engine on CPU: 23 fps
Subdivision level 3 (800k triangles):
- Softimage: 2 fps
- Maya SmoothProxy: 2.5 fps
- Fabric Engine on CPU: 6 fps
It is interesting to note that all three of these implementations are multithreaded; Fabric Engine’s 2 to 6 times speed increase does not come solely from multithreading, but rather from efficient data structures, and from a computation model that allows for very efficient use of computation resources. It is also interesting to note that while such information is not available, it is reasonable to assume that Fabric Engine’s implementation of subdivision surfaces in our high-level KL language is both simpler and smaller than in these software packages.

Using the GPU for even greater performance

The previous section already shows a very significant 2x to 6x performance edge for Fabric Engine, but more still can be done. While we are progressing quickly towards targeting GPUs directly with the KL compiler, the subdivision surfaces problem lends itself well to targeting the GPU manually. By storing precomputed data in textures, and by performing the evaluation of the subdivided surface in a vertex shader, more performance still was easily obtained. Here are our results again, this time including our GPU version of the code:

Subdivision level 1 (50k triangles):
- Softimage: 10 fps
- Maya SmoothProxy: 28 fps
- Fabric Engine on CPU: 60 fps
- Fabric Engine + GPU: 273 fps
Subdivision level 2 (200k triangles):
- Softimage: 5 fps
- Maya SmoothProxy: 9 fps
- Fabric Engine on CPU: 23 fps
- Fabric Engine + GPU: 95 fps
Subdivision level 3 (800k triangles):
- Softimage: 2 fps
- Maya SmoothProxy: 2.5 fps
- Fabric Engine on CPU: 6 fps
- Fabric Engine + GPU: 29 fps
Fabric Engine, when using the GPU for evaluating the surface points, therefore has a whopping 10x to 27x performance advantage over traditional content creation tools when evaluating subdivision surfaces.

The following video shows our results both for CPU and GPU implementations:

Conclusion and Future Work

Catmull-Clark subdivision surfaces were implemented in the Fabric Engine Creation Platform, using Fabric Engine’s Map-Reduce framework to carry its computation. Results show a 2 to 6 times frame rate increase in evaluating a subdivision surfaces, compared to 2 leading commercial content creation tools. We have also shown that by performing the evaluation on the surface in a vertex shader on the GPU, an additional 4 to 5x speed gain is achieved, for a total speedup of 10x to 27x. Future efforts will allow KL programs to compile directly to GPU, foregoing the need for the user to map their problems to shaders and allowing them to target the GPU and CPU indifferently and automatically.

Comments

Like

Add New Comment

Type your comment here.

Showing 1 comment

Simone De Marzo
Two things:

1 - when you played the 4th subdivision level on the gpu mode, it was written 50/60 fps and you say it's playing 50 to 60 fps, but it's clearly not true, they seem to me 5 fps. The 3th level of subs, was really fluid and believable as 60fps, the 4th is not.

2 - the softimage values are always better in the video, then the maya one. But it is written the contrary in this page.

4 months ago