Level of Detail for "Tumbler" software - Michael Ferguson Fall 2002

Abstract:

I investigate level of detail methods for an animation project. I begin by explaining several level of detail methods, and by describing my project requirements, and then I investigate freely available level of detail implementations. Finally, I compare the two most applicable methods: Turk's *plycrunch* (cell collapse) and Garland's *propslim* (vertex-pair collapse with a quadric error metric).

Survey of Methods (as described in [1]):

Level of Detail methods fall into two categories: those which repeatedly apply a primitive simplification operator, such as collapsing two vertices, and those which must be applied to an entire model at once. I will describe the primitive simplification operators, detail the general algorithm used with these operators, and outline a few other algorithms.

Simplification Operators:

Edge-Collapse

The edge collapse operator takes two vertices connected by an edge and identifies them, thereby reducing a mesh by two triangles, as Figure 1 shows. So an edge-collapse operator is a function taking two vertices $v1$ and $v2$ and identifying them into a new vertex, w; it could be written as $collapse(v1, v2) = w$.

Edge-collapse operators include half-edge collapse, in which the position of one of the original vertices is taken as the position of the resulting vertex (i.e. $collapse(v1, v2) = v1$ or $v2$), and full edge-collapse, in which new point w is placed in-between $v1$ and $v2$, possibly to reduce some error function. Edge collapse must be used carefully to prevent the introduction of visual artifacts from mesh foldovers. In Figure 2, for example, collapsing the red edge creates a mesh in which the blue triangle has flipped orientation and folded over itself. Edge collapse algorithms usually use a heuristic to prevent mesh foldovers [1,8].

Vertex-Pair Collapse

The vertex-pair collapse operator takes two nearby vertices which may or may not be connected by an edge and identifies them. This operator only reduces the triangle count of the model if the vertices are connected by an edge. As Figure 1 shows, the vertex-

pair collapse algorithms can close holes and connect disconnected parts of a model. However, a threshold distance must be chosen to make the algorithm computationally effective [8].

Triangle Collapse

With triangle collapse, an entire triangle is selected and all of its vertices are identified. This operator can be expected to remove four triangles at a time. In Figure 1, each of the red edges is an edge for both the red triangle and also for an adjacent. When the three red points are identified, these three triangles are removed in addition to the selected triangle. It's important to note that a triangle collapse can be described as two edge collapses, and so edge-collapse can be viewed as a finer-resolution version of this method [1].

Vertex Removal

Vertex removal operates by simply removing one vertex at a time and then re-triangulating the area that had triangle edges connected to the vertex. In Figure 1, for example, the vertex *v* is removed and the surrounding vertices are re-triangulated with the purple edges. Since there are several ways to retriangulate the resulting area, this algorithm usually solves a discrete optimization problem to produce a good simplification[1,8].

General Algorithm & Error Metrics:

The algorithms which have a primitive simplification operator share a general algorithm [1]. The algorithm for edge collapse would start by putting all pairs of vertices connected by an edge into a data structure. Then the simplification program removes the pair which, when contracted, will induce the smallest error to the model. Next the program applies the simplification operator - in this case, it collapses the edge and replaces references to each of the two vertices with references to the new vertex. Note that in this general algorithm, the error metric determines the order in which the operations are applied. As Garland describes, several error metrics exist, including measuring the distance to a plane approximating the surrounding vertices, measuring the distance to the actual model, and using heuristics such as edge length, local curvature, or dihedral edge angle [8]. Because the former methods are computationally expensive, and the latter are unreliable, Garland suggests quadric error metrics [6,8]. I will describe Garland's conceptual model for this error metric. Suppose that we are collapsing two vertices, v_1, and v_2. Take the set S_1 to be the set of planes determined by faces containing v_1, and S_2 to be the planes from the faces containing v_1. Now in placing the new vertex w, minimize

the sum-squares distance to the planes in the set $S_1 \cup S_2$. In three dimensions, the surfaces of constant error measured in this way are quadric surfaces, and so the technique is called the quadric error metric. In practice, only one error metric needs to be associated with each vertex, and the union described is implemented by a sum of quadrics [8].

Algorithms without Operators:

Cell Collapse

In cell collapse methods, a grid of some sort is imposed on the model to divide it into a number of cells. Then all the vertices in that cell are identified into a single vertex, and the new model is created by reconnecting areas that were previously connected [1]. Figure 2 is a two-dimensional example. The squares are the cells, and the red dots are the average coordinate values of the vertices in each cell. So in this case, $w = (v_1 + v_2)/2$. Next, in every edge (v_1,v_2) the edge is replaced by (w_1,w_2), where w_1 and w_2 are the representative vertices for the cells containing v_1 and v_2, respectively. All edges of the form (w, w) are removed because they are degenerate. Cell collapse methods operate very quickly, but they may have low quality [8].

Volume Processing & □-hull methods

Volume processing methods first convert the model into a volumetric grid, then they simplify this grid, and finally they recompute a triangle mesh. □-hull methods operate by using a series of spheres or planes to carve out a simplified model [1]. Both the volume processing and the □-hull methods make sense as purely topological simplifications. However, they do not seem to be able to reduce the color-based error of the simplified model. Furthermore, the □-hull method can introduce bumps in the simplified model, and the volume processing methods would be difficult to implement if the original models are polygonal.

RSimp

RSimp (for Reverse Simplification) builds a very coarse approximation for a model and then refines it. For this reason, it can rapidly simplify large data sets. *RSimp* uses clusters of triangles; it is similar to cell collapse, but these triangle clusters are not necessarily three-dimensional cubes. *RSimp* begins by creating 8 initial clusters and then splitting the clusters with the most error from the original surface as needed to meet a vertex or error bound.

Requirements:

The animation software will use models made up of triangles with color data at each vertex. Users of the software will interactively specify a composition of different effects that can be applied to a model (such as opening up a model). The program will process these effects by applying transformations to models in order, where each transformation computes new triangle data based on input triangle data and parameters. These parameters for transformations are to be specified interactively and graphically, and so previews of transformations must be renderable at interactive speeds. For this reason, I will need to use a level of detail algorithm to simplify the input models. Currently, expected input has from 16,000 to 311,000 faces. The level of detail algorithm will need to:

1) take any triangle data (including non-manifold surfaces) as input
2) output triangle data (and not custom data structures)
2) create qualitatively similar simplified models
3) run at a reasonable speed (take less than a few minutes per model)
4) create color data for the simplified model

Assessment of Available Methods:

I searched for freely available model simplification programs. I found *plycrunch, RSimp, plysimplify, qslim*, and *propslim*. Although all of these methods create simplified polygonal models (i.e., none of them are volumetric or view-dependent), only *plycrunch* and *propslim* are immediately applicable to my animation software.

Neither *RSimp* nor *plysimplify* meet my requirements. Although *RSimp* can work with large models very quickly, it does not store color information in the simplified model [4]. *plysimplify* is Cohen et al.'s implementation of simplification envelopes [3]. This method uses the vertex removal operator is used and an error metric which measures the distance from the model to the simplification and from the simplification to the model. However, this method requires that the input model be manifold, which is not true in my case; in fact my models contain large holes (see Figure 5).

plycrunch [2] is Turk's implementation of Rossignac's method [7] of a cell collapse in which the representative vertex for a cell is simply the average of the vertices in that cell. *plycrunch* preserves color data by copying the color information from one of the existing vertices in each cell to the representative vertex for that cell. *plycrunch* meets all four of my requirements.

qslim and *propslim* are both implementations of vertex-pair collapse using

Garland's quadric error metric [8,9] as I discussed above. *propslim* optimizes color information in each new vertex in addition to xyz coordinates [8,9]. Although *qslim* is not suitable because it does not create color data in the simplified model, *propslim* meets my requirements.

Comparison of *plycrunch* and *propslim*:

While *plycrunch* and *propslim* both meet my four requirements, I find that although *plycrunch* is faster, *propslim* creates better models.

Performance

I measured the time it took each of these two programs to simplify different input models. All time measurements were made on a PowerMac G4/867 MHz with 650 MB

	Original Number of Triangles	Reduced Number of Triangles	*plycrunch* Running Time (seconds)	*propslim* Running Time (seconds)
head.ply	310,924	3,571	2.44	59.84
billc1.ply	168,606	3,535	1.30	28.51
billk1.ply	209,609	2,258	1.62	36.02

RAM running Mac OS X 10.2.2; the times in the table are user times as *time* reports.

As this table demonstrates, *plycrunch* only takes a second or two to run, while *propslim* takes about twenty times as long. Still, both programs simplify my files in under two minutes.

Quality of Output

Figures 4-8, show images rendered from the original model and the model simplified by these two methods. In general, the *plycrunch* output looks worse than the *propslim* output. For example, *plycrunch* smears features of the face in Figure 1 and renders the headband indiscernible. I believe that these problems are due to the arbitrary color value *plycrunch* uses for each representative vertex and also due to the fact that *plycrunch* does not optimize its choices of representative vertices.

Furthermore, *plycrunch* has a tendency to arbitrarily amplify and smooth over holes in the data. In Figure 5, for example, the right leg of the cross-legged figure is has an unusual hole behind the knee that was not present in the original. Furthermore, the holes on the top of the shoulders seem to get larger. Looking at the left arm of the model in Figure 7, one can see again where *plycrunch* fused the arm and the body and where it

introduced a hole. Lastly, in Figure 8, *plycrunch* expands the hole in the bellybutton of the figure into a large square.

The only drawback to *propslim*, besides its longer running time, is that it creates distortions on the boundary of the original model in some cases. In Figure 7, for example, the arms and legs have unusual wavy boundaries that were not present in the original model. I believe that these distortions may be the result of rapidly varying color information in those areas of the model. However, as Figure 9 shows, when I simplify the model without the color data, the model still has these wavy sections. Perhaps these irregular areas come from incorrect boundary handling in *propslim*.

Conclusion

Although neither *plycrunch* nor *propslim* are without problems, I prefer *propslim* because it creates much higher quality models with the same number of triangles as a *plycrunch* output. However, in the event that my level of detail method needs to be executed often, I will use *plycrunch* instead because of its rapid execution time.

Future Work

Future work could investigate the boundary problem with *propslim* and try to address the problem. An algorithm similar to *plycrunch* could simplify more than once using different cell pavings to reduce the arbitrary expanding and smoothing of holes. Finally, *plycrunch* could probably make much better-looking output if it averaged the color values in a cell to make the color for the representative vertex instead of arbitrarily choosing a color from one of the cell vertices.

Works Cited:

[1] David Luebke, Martin Reddy, Jonathan Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for 3D graphics*, pages 3-83, 2003.

[2] Greg Turk. *plycrunch*. Computer software and source code, available at http://www.cs.unc.edu/~geom/Powerplant/zips/ply_utilities.zip

[3] Jonathan Cohen, Varshney Amitabh , Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification Envelopes. In *SIGGRAPH '96 Proc*., pages 119-128, Aug. 1996.

[4] Dmitry Brodsky and Benjamin Watson. *RSimp*. Computer software, available at http://www.cs.nwu.edu/~watsonb/school/projects/rsimp/index.html

[5] Dmitry Brodsky, Benjamin Watson. *Model simplification through refinement*. In *Proc. Graphics Interface* (Montreal, May), pages 221-228, 2000.

[6] Michael Garland and Paul Heckbert. *Surface Simplifcation Using Quadric Error Metrics*. In *SIGGRAPH 97 Proc.,* pages. 209-216. 1997.

[7] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications,* pages 455-465, 1993.

[8] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. Ph.D. Thesis, pages 35-45, 56. Available at http://graphics.cs.uiuc.edu/~garland/research/thesis.html

[9] Michael Garland. *qslim 2.0*. Computer software and source code, available at http://graphics.cs.uiuc.edu/~garland/software/qslim.html

[10] Jonathan Cohen, Marc Olan, and Dinesh Manocha. *Appearance-Preserving Simplification*. In *SIGGRAPH '98 Proc*., July 1998.