Mesh Refinement based on Euler Encoding

Le-Jeng Shiue
University of Florida
sle-jeng@cise.ufl.edu

Jörg Peters
University of Florida
jorg@cise.ufl.edu

Abstract

A sequence of mesh manipulations that preserves the Euler invariant is called an Euler encoding. We propose new, efficient Euler encodings for primal and dual mesh refinement. The implementations are analyzed and compared to array-based, connectivity-free refinement and to reconstruction of the refined mesh.

1. Introduction

Euler operators reconfigure the neighborhood of a mesh node while preserving the structural invariant of the Euler count. They are the standard atomic operation for editing meshes [14]. Euler operators are often realized as updating adjacency pointers of the mesh data structure (such as the halfedge data structure [22]). A subset of Euler operators are shown in Figure 4. Mesh algorithms such as progressive mesh [9], mesh remeshing [1, 7], mesh compression [16], and mesh refinement are based on a sequence of Euler operators to assure the combinatorial integrity of the transformed mesh. The corresponding sequence of operation is called an Euler encoding.

Mesh refinement is a core operation for many computational and graphics applications, including subdivision algorithms (see e.g. [21]) and multiresolution modeling. Figure 1 illustrates four common refinement schemes. Figure 2 shows a mesh sequence generated by a PQQ-based subdivision algorithm where new nodes are placed as weighted averages of old ones to smooth the polyhedron. The local neighborhood of the old nodes is called stencil, and the correspondence between the new node and its stencil is called stencil correspondence. An Euler encoding of mesh refinement is efficient if it (i) minimizes the operation count, and (ii) encodes updates (i.e. stencil correspondences) without the need for extraneous data structures, such as tagging vertices. The encoded updates are crucial in implementing a generic library of mesh refinement on a generic mesh data structure [19].

Figure 1: Refinement schemes (initial mesh top, refined mesh bottom): primal quadrilateral quadrisection (PQQ), primal triangle quadrisection (PTQ), dual quadrilateral quadrisection (DQQ) and $\sqrt{3}$ triangulation.

Figure 2: Refinement of a rook model by Catmull-Clark subdivision [5]. Nodes of the refined mesh are placed according to geometry rules that depend on the local neighborhood graph.

This paper focuses on Euler-encoding primal and dual quadrisection refinements. Dual schemes are often considered incompatible with Euler encoding, but, in fact, we exhibit three new, alternative Euler encodings for the dual quadrilateral quadrisection! The performance effect of using Euler operations is evaluated by comparing the encoding of the PQQ refinement with an optimized array-based implementation which is connectivity-free. The three Euler encodings of the DQQ refinement are compared and analyzed, and we establish the advantage of the Euler encoding over another type of approach, namely reconstructing the refined mesh.
2. Mesh Data Structures and Euler Operations

A mesh is a graph whose primitives (i.e. vertices, edges and facets) carry attribute information such as vertex positions or facet colors. The topology of a mesh is represented by a combinatorial manifold structure. A mesh \( M = \{ F, E, V \} \), where \( F \), \( E \) and \( V \) are sets of facets, edges and vertices. A facet is an ordered set of vertices \( (f = \{ v_0, v_1, ..., v_{n-1} \}, n \) is the valence), and an edge is a vertex pair \( (e = \{ v_0, v_1 \}) \). A mesh is 2-manifold if every inner point has a neighborhood homeomorphic to a disk (or to a half-disk on the mesh boundary). In this paper, a mesh is always a 2-manifold mesh.

The most popular data structures for manipulating meshes are edge-based, i.e. use the adjacency of edges to represent the connectivity. Examples of edge-based mesh data structures are the winged-edge data structure \([2]\), the halfedge data structure \([22]\), and the quadedge data structure \([8]\). Adjacency of primitives are usually realized as pointers (called adjacency pointers), and low level mesh editing (such as adding or deleting edges) amount to updating adjacency pointers. The halfedge data structure best combines simplicity and flexibility and comes in two flavors (Figure 3). Each halfedge has an opposite pointer to its paired halfedge, pointers to the previous and the next halfedges of the incident facet (or vertex), and pointers to the incident facet and vertex. Depending on the regularity of the mesh some adjacency pointers can be omitted \([4]\). The Computational Geometry Algorithm Library (CGAL) represents polyhedron meshes (Polyhedron,3) \([10]\) based on the halfedge data structure.

Euler operations change the mesh and, by the same time, preserve the combinatorial integrity represented by the Euler characteristic \( V - E + F = 2 - 2g \) where \( V \), \( E \), and \( F \) are respectively the numbers of vertices, edges, and facets and \( g \) is the genus. Any two meshes homeomorphic to each other have the same Euler characteristic and can be transformed into each other through a sequence of Euler operations. Some commonly-used Euler operations are shown in Figure 4 and Figure 5. The number of the pointer updates of a Euler operation is summarized in Table 1. The number is based on a halfedge data structure with the complete set of the adjacency pointers (i.e. 3 pointers to adjacent halfedge, 1 each to the incidental facet and vertex.)

One special subset of Euler operations is the Stellar operations \([12]\). Stellar operations applies to simplicial complexes, i.e. triangulated 2-manifolds. The flip-edge is called Stellar move, and the breakFacet is called Stellar subdivision. Stellar operations have been used in mesh refinements on a triangulated mesh. Because Stellar operations are re-
and illustrated in Figure 3). Euler encodings of primal schemes only involve vertex and edge insertions. The Euler encoding of PQQ refinement requires more steps as shown in Figure 6: The array-based implementation of the PQQ refinement. (top) refined mesh lines, (bottom) layout of nodes. amounts to scaling the vertex array.

**Listing 1:** Euler encoding of PQQ refinement (see Figure 7).

```latex
\begin{align*}
\textbf{INPUT: } & \text{ initial mesh } \mathcal{M} = \{\mathcal{F}, \mathcal{E}, \mathcal{V}\} \\
\textbf{OUTPUT: } & \text{ PQQ-refined mesh } \\
& \mathcal{V}'_{\mathcal{F}} := \{\} \\
& \text{ for all } \mathcal{e} \in \mathcal{E} \text{ do} \\
& \quad \mathcal{v}_{\mathcal{e}} \leftarrow \text{insertVertex}(\mathcal{e}) \quad \ldots \quad (a) \\
& \quad \mathcal{V}'_{\mathcal{E}} := \mathcal{V}_{\mathcal{E}} \cup \mathcal{V}_{\mathcal{V}} \\
& \text{ end for} \\
& \text{ for all } \mathcal{f} \in \mathcal{F} \text{ do} \\
& \quad \mathcal{V}'_{\mathcal{F}} := \mathcal{V}_{\mathcal{F}} \cup \mathcal{V}_{\mathcal{F}} \\
& \quad \mathcal{e} \leftarrow \text{splitCorner}(\mathcal{f}, \mathcal{v} \cap \mathcal{V}_{\mathcal{F}}) \quad \ldots \quad (b) \\
& \quad \mathcal{v}_{\mathcal{f}} \leftarrow \text{insertVertex}(\mathcal{e}) \quad \ldots \quad (c) \\
& \quad \text{insertEdge}(\mathcal{v}_{\mathcal{f}}, \mathcal{V}_{\mathcal{F}} - \mathcal{e}) \quad \ldots \quad (d) \\
& \text{ end for}
\end{align*}
```

Here \(f\) is the set of vertices that define the facet.

Mesh refinement recursively reconfigures the initial mesh by dividing facets (e.g. PQQ refinement) or splitting vertices (e.g. DQQ refinement). Uniform facets or vertices are recursively generated in the process. Figure 1 shows four regular refinement schemes. A mesh refinement is called primal if we can naturally associate a vertex of refined mesh with each vertex of the original mesh. Euler encodings of primal schemes only involve vertex and edge insertions. The Euler encoding of \(\sqrt{3}\) triangulation can be expressed as inserting a barycentric vertex in each facet and then flipping every old edges. The Euler encoding of PQQ refinement requires more steps as shown in Figures 7 and 8.

3. Euler Encoding of Mesh Refinement

Mesh refinement recursively reconfigures the initial mesh by dividing facets (e.g. PQQ refinement) or splitting vertices (e.g. DQQ refinement). Uniform facets or vertices are recursively generated in the process. Figure 1 shows four regular refinement schemes. A mesh refinement is called primal if we can naturally associate a vertex of refined mesh with each vertex of the original mesh. Euler encodings of primal schemes only involve vertex and edge insertions. The Euler encoding of \(\sqrt{3}\) triangulation can be expressed as inserting a barycentric vertex in each facet and then flipping every old edges. The Euler encoding of PQQ refinement requires more steps as shown in Figures 7 and 8.

**Listing 2:** Splitting encoding of DQQ refinement (see Figure 8).

```latex
\begin{align*}
\textbf{INPUT: } & \text{ initial mesh } \mathcal{M} = \{\mathcal{F}, \mathcal{E}, \mathcal{V}\} \\
\textbf{OUTPUT: } & \text{ DQQ-refined mesh } \\
& \mathcal{V}'_{\mathcal{F}} := \{\} \\
& \text{ for all } \mathcal{v} \in \mathcal{V}_{\mathcal{F}} \text{ do} \\
& \quad \text{splitStar}(\mathcal{v}) \\
& \quad \text{weldFacet}(\mathcal{v}) \\
& \text{ end for} \\
& \text{ for all } \mathcal{e} \in \mathcal{E} \text{ do} \\
& \quad \text{deleteEdge}(\mathcal{e}) \\
& \text{ end for}
\end{align*}
```

Listing 1 and illustrated in Figure 7. The PQQ Euler encoding (a) inserts a (edge-)vertex on each edge, then, for each facet, (b) an edge is inserted between two neighbor edge-vertices, (c) a (facet-)vertex is added on that edge, and (d) edges are inserted to connect all edge-vertices of the facet. In a generic mesh data structure with no specific tag to indicate the type (i.e. the stencil correspondences such as edge-vertex and facet-vertex) of the vertex in refined mesh, the order of the Euler operations is used to determine the stencil correspondences of a refined vertex [17].

Encoding dual refinement is more tricky since the initial mesh is not retained, and even trickier to maintain the stencil correspondences without tagging the vertex. Figure 8 shows three Euler encodings of the DQQ refinement. Factoring decomposes the DQQ refinement into two mid-edge refinements. A midedge refinement inserts a vertex on each edge, connects the new vertices within a facet, and then deletes the initial vertices. Loop and \(\sqrt{3}\) subdivision in OpenMesh [3] are similarly factored [20]. The splitting encoding generates, for each vertex, new nodes, corresponding to the corners of the incident facets. Both the factoring and splitting generate intermediate vertices that are deleted in the end of the encoding. The number of intermediate ver-
Table 2: Refinement (in seconds) and pointer updates (in parentheses) of Doo-Sabin subdivision on the tripod and the gripper model. “Rebuild” indicates the implementation based on the reconstruction of the refined mesh.

Table 3: Speed improvement and reduction of the Euler encoding of Tilting vs Splitting. SI indicates the speed improvement, and LR indicates the length reduction.

where $I_{v,e}$ and $D_e$ indicate the number of the updated pointers for each insertVertex, insertEdge and deleteEdge; $W_f$ and $B_f$ indicate the number for each weldFacet and breakFacet; $S_v$ and $F_e$ indicate the number for each splitStar and fillStar. Table 2 summarizes the splitting and tilting encoding: the length is reduced by 39.4% and the speed is improved by 20.9% in the fifth refinement on the gripper mesh (see Table 3). Table 2 also compares with mesh reconstruction based on the incremental builder and modifier callback mechanism of CGAL::Polyhedron_3. Our Euler encoding is almost 14 times faster after five refinements on the tripod mesh.

Euler operations vs array-based mesh refinement. Array-based implementation takes advantages of the regularity of the quadrisection of a quadrilateral: since the new vertices are arranged in a 2D array the connectivity is implied by the grid and simple index calculation. Nodes on boundaries of arrays need to be replicated and there is overhead for collecting and distributing data for (irregular-)corners of the array. This overhead makes array-based refinement slower in the first steps of the PQQ refinement. The interesting question is, after how many steps the cost of applying the Euler operations to a refined mesh exceeds this overhead. Table 4 shows that only after three steps array-based implementation, where applicable, is superior.

Comparison with OpenMesh. To gauge the efficiency of our implementation, we compare with the OpenMesh im-

### Table 2: Refinement (in seconds) and pointer updates (in parentheses) of Doo-Sabin subdivision on the tripod and the gripper model. “Rebuild” indicates the implementation based on the reconstruction of the refined mesh.

<table>
<thead>
<tr>
<th>Subdivision Step</th>
<th>tripod</th>
<th>Gripper</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Splitting</td>
<td>Tilting</td>
</tr>
<tr>
<td>1</td>
<td>0.0007 (4484)</td>
<td>0.0006 (2720)</td>
</tr>
<tr>
<td>2</td>
<td>0.0026 (18016)</td>
<td>0.0022 (10956)</td>
</tr>
<tr>
<td>3</td>
<td>0.0101 (71800)</td>
<td>0.0084 (438084)</td>
</tr>
<tr>
<td>4</td>
<td>0.0564 (2891944)</td>
<td>0.0471 (1753596)</td>
</tr>
<tr>
<td>5</td>
<td>0.1939 (11569096)</td>
<td>0.1594 (695892)</td>
</tr>
</tbody>
</table>

### Table 3: Speed improvement and reduction of the Euler encoding of Tilting vs Splitting. SI indicates the speed improvement, and LR indicates the length reduction.

<table>
<thead>
<tr>
<th>Subdivision</th>
<th>tripod</th>
<th>Gripper</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI (%)</td>
<td>LR (%)</td>
<td>SI (%)</td>
</tr>
<tr>
<td>1</td>
<td>14.2857</td>
<td>39.3399</td>
</tr>
<tr>
<td>2</td>
<td>15.3817</td>
<td>39.1874</td>
</tr>
<tr>
<td>4</td>
<td>0.0564</td>
<td>0.0471</td>
</tr>
<tr>
<td>5</td>
<td>17.7927</td>
<td>39.3548</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Subdivision</th>
<th>Loop subdivision</th>
<th>√3 subdivision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CGAL</td>
<td>OpenMesh</td>
</tr>
<tr>
<td>1</td>
<td>0.0013</td>
<td>0.0284</td>
</tr>
<tr>
<td>2</td>
<td>0.0215</td>
<td>0.1161</td>
</tr>
<tr>
<td>3</td>
<td>0.0226</td>
<td>0.4672</td>
</tr>
<tr>
<td>4</td>
<td>0.1173</td>
<td>1.8459</td>
</tr>
<tr>
<td>5</td>
<td>0.4529</td>
<td>7.5003</td>
</tr>
</tbody>
</table>

Table 5: Subdividing the triangulated rook mesh. All statistics are in seconds.

Implementation of Loop and √3 subdivisions which are based on factoring. Table 5 reflects the fact that factoring the refinement introduces an additional connectivity and geometry overhead; and geometry operations are usually computational expensive.

5. Conclusion

Euler operators are standard atomic editing operations on meshes and their optimization is crucial for the software performance. This paper introduced new, efficient Euler encodings for PQQ and DQQ refinements on a generic mesh data structure. For the limited set where it applies, array-based refinement outperforms Euler encoding after the first few refinements, but rebuilding the mesh from scratch is shown to be slower than Euler encoding.

References

Figure 7: An Euler encoding of the PQQ refinement (see Listing 1). Light blue indicates the insertion of the new vertex and edge.

Figure 8: Three alternative Euler encodings of the DQQ refinement (see Listing 2 and Listing 3). Light blue indicates the insertions of the vertex and the edge.

Figure 9: Test models of the Euler encoding: tripod, rook, and gripper rendered with Catmull-Clark subdivision.