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].

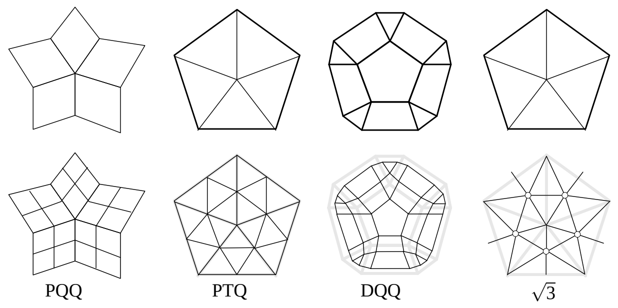


Figure 1: Refinement schemes (initial mesh *top*, refined mesh *bottom*): primal quadrilateral quadrisection (PQQ), primal triangle quadrisection (PTQ), dual quadrilateral quadrisection (DQQ) and $\sqrt{3}$ triangulation.

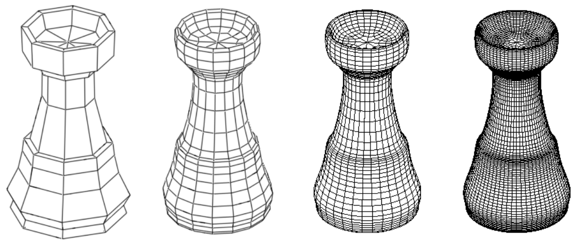
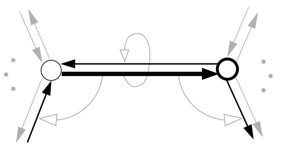


Figure 2: Refinement of a rook model by Catmull-Clark subdivision [5]. Nodes of the refined mesh are placed according to geometry rules that depend on the local neighborhood graph.

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.



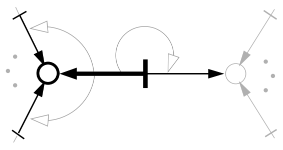


Figure 3: The two flavors of the halfedge mesh data structure: each halfedge points to a *previous*, a *next* and an *opposite* halfedge.

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 $\mathbb{M} = \{\mathbb{F}, \mathbb{E}, \mathbb{V}\}$, where $\mathbb{F}$, $\mathbb{E}$ and $\mathbb{V}$ 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 up-

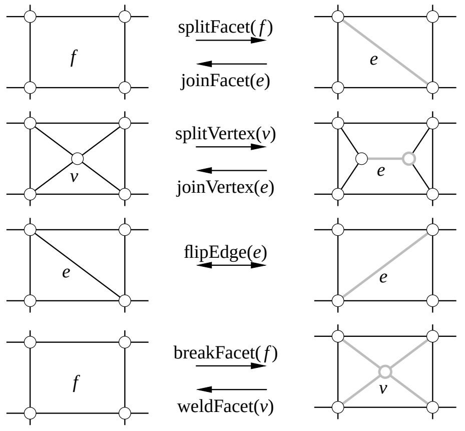


Figure 4: Commonly used examples of Euler operations. Gray indicates the updated edges and vertices.

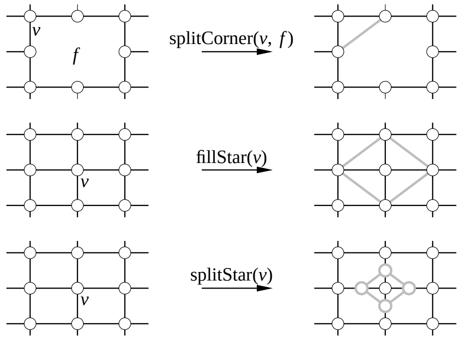


Figure 5: Additional examples of Euler operations.

join/split facet	join/split vertex	flip edge	break/weld facet	split star
14	15	14	$12n+1$	$15n$

Table 1: Number of pointer updates for Euler operations (see Figures 4 and 5) using the complete linked halfedge data structure. Here n is the degree of the facet or the vertex.

dates 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-

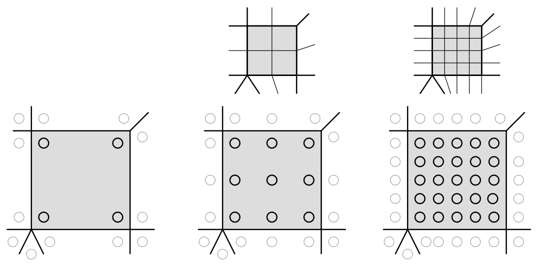


Figure 6: The array-based implementation of the PQQ refinement. (*top*) refined mesh lines, (*bottom*) layout of nodes. amounts to scaling the vertex array.

```
INPUT: initial mesh $\mathbb{M}=\{\mathbb{F},\mathbb{E},\mathbb{V}\}$
OUTPUT: PQQ-refined mesh
  $\mathbb{V}_e := \{\}$
  for all $e \in \mathbb{E}$ do
    $v_e \leftarrow$ insertVertex($e$) ..... (a)
    $\mathbb{V}_e := \mathbb{V}_e \cup v_e$
  end for
  for all $f \in \mathbb{F}$ do
    $\mathbb{V}_f := \mathbb{V}_e \cap f$
    $e \leftarrow$ splitCorner($f$, $v \in f \cap \mathbb{V}$) ..... (b)
    $v_f \leftarrow$ insertVertex($e$) ..... (c)
    insertEdge($v_f, \mathbb{V}_f - e$) ..... (d)
  end for
```

Listing 1: Euler encoding of PQQ refinement (see Figure 7). Here f is the set of vertices that define the facet.

stricted to triangles, a special data structure and a preprocessing step are required to support general meshes. Euler operations is more flexible in supporting general meshes and various refinement schemes.

To implement mesh refinements, array-based data structures [15, 18] represent an alternative to edge-based data structures, when the mesh is limited to quadrilateral and triangle meshes. Figure 6 describes the process of the PQQ refinement of a quadrilateral mesh. Here, the adjacency relations of the array are implicit (hence no adjacency pointers), and refinement scales the underlying array.

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 the $\sqrt{3}$ 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

```
INPUT: initial mesh $\mathbb{M}=\{\mathbb{F},\mathbb{E},\mathbb{V}\}$
OUTPUT: DQQ refined mesh
  $\mathbb{V}_f := \{\}$
  for all $f \in \mathbb{F}$ do
    $v_f \leftarrow$ breakFacet($f$)
    $\mathbb{V}_f := \mathbb{V}_f \cup v_f$
  end for
  for all $v \in \mathbb{V}_f$ do
    splitStar($v$)
    weldFacet($v$)
  end for
  for all $e \in \mathbb{E}$ do
    deleteEdge($e$)
  end for
  for all $v \in \mathbb{V}$ do
    fillStar($v$)
    weldFacet($v$)
  end for
```

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

```
INPUT: initial mesh $\mathbb{M}=\{\mathbb{F},\mathbb{E},\mathbb{V}\}$
OUTPUT: DQQ refined mesh
  $\mathbb{V}_f := \{\}$
  for all $v \in \mathbb{V}$ do
    splitStar($v$)
    weldFacet($v$)
  end for
  for all $e = \overrightarrow{v_1 v_2} \in \mathbb{E}$ do
    splitCorner($v_1$, $f : \overrightarrow{v_1 v_2} \in f$)
    splitCorner($v_2$, $f : \overrightarrow{v_2 v_1} \in f$)
    deleteEdge($e$)
  end for
```

Listing 3: Tilting encoding of DQQ refinement (see Figure 8).

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 midedge 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-

Subdivision Step	tripod					gripper			
	Splitting		Tilting		Rebuild	Splitting		Tilting	
1	0.0007	(4484)	0.0006	(2720)	0.0006	0.0261	(180764)	0.0212	(109616)
2	0.0026	(18016)	0.0022	(10956)	0.0028	0.1275	(722656)	0.1034	(438084)
3	0.0101	(71800)	0.0084	(43572)	0.0214	0.4820	(2891944)	0.3902	(1753596)
4	0.0564	(286936)	0.0471	(174036)	0.2748	1.8639	(11569096)	1.5577	(7015644)
5	0.1939	(1147480)	0.1594	(695892)	7.4789	7.6093	(46277704)	6.0163	(28063836)

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.

tices in factoring is the number of initial edges. The number of intermediate vertices in splitting is the number of initial facets. Since the number of edges is larger than the number of facets, splitting has a smaller number of connectivity updates than factoring. In a quadrilateral mesh, splitting creates only half intermediate vertices of factoring. However, we can do even better. The *tilting* avoids all intermediate vertices by leveraging that *each new vertex corresponds to a halfedge*. The algorithms of the DQQ refinement based on splitting and tilting encodings are listed in List 2 and 3 and illustrated in Figure **??**.

4. Implementation and Performance Analysis

The PTQ, PQQ, DQQ and $\sqrt{3}$ refinements were realized as Euler encodings on `CGAL::Polyhedron_3` based on the generic programming paradigm and static binding [1]. We implemented Loop [13], Catmull-Clark, Doo-Sabin [6] and $\sqrt{3}$ [11] subdivisions of the test meshes shown in Figure 9. The tripod contains 20 vertices, 18 facets, and 36 edges; the rook contains 130 vertices, 256 triangles, and 384 edges; the gripper contains 716 vertices, 726 facets, and 1452 edges. Mesh refinements in general triple or quadruple the mesh size by each refinement. For example, one step PQQ refinement on the gripper mesh creates 2894 vertices, 2904 facets, and 5808 edges. Test programs are compiled with GCC 3.3.2 and run on an Intel Pentium4 2.4GHz with 1GB RAM.

Comparison of three alternative Euler encodings of the DQQ refinement introduced in Figure 8. Here, we count the number of the pointer updates on a halfedge data structure. Assume that the initial mesh contains n_f quadrilaterals, n_e edges and n_v vertices. To simplify the formulas, the average valence is also assumed to be four. Then the following table lists the *length L of each Euler encoding*. The length is the total number of the pointer updates in the encoding.

$$\begin{aligned} L_{factor} &= (n_e+4n_f)I_v+4(2n_f+n_v)I_e+(n_v+n_e)W_f \\ L_{split} &= n_fB_f+n_fS_s+(n_f+n_v)W_f+n_eD_e+n_vF_s \\ L_{tilt} &= n_vS_s+n_vW_f+2n_eI_e+n_eD_e \end{aligned}$$

1 http://www.cgal.org/

subdivision	tripod		gripper	
	SI (%)	LR (%)	SI (%)	LR (%)
1	14.2857	39.3399	18.7739	39.3596
2	15.3846	39.1874	18.9020	39.3786
3	16.8317	39.3148	19.0456	39.3627
4	16.4894	39.3468	16.4279	39.3588
5	17.7927	39.3548	20.9349	39.3578

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_s and F_s 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-

subdivision	tripod		gripper	
	Array	Halfedge	Array	Halfedge
1	0.0004	0.0003	0.0157	0.0094
2	0.0013	0.0010	0.0585	0.0413
3	0.0043	0.0038	0.1810	0.2139
4	0.0156	0.0155	0.6595	0.8313
5	0.0621	0.0718	2.6196	3.4718

Table 4: Array-based and halfedge-based Catmull-Clark subdivision on the tripod and the gripper model. At refinement level 5, array-based implementation is 30% faster than Euler-based halfedge implementation on the gripper model. All statistics are in seconds.

Subdivision	Loop subdivision		$\sqrt{3}$ subdivision	
	CGAL	OpenMesh	CGAL	OpenMesh
1	0.0013	0.0284	0.0022	0.0230
2	0.0215	0.1161	0.0069	0.0712
3	0.0226	0.4672	0.0270	0.2101
4	0.1173	1.8459	0.1235	0.6693
5	0.4529	7.5003	0.3777	1.8574

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

plementation of Loop and $\sqrt{3}$ 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

[1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing. In *SIGGRAPH '03 Conference Proceedings*, pages 485–493, 2003.

[2] B. G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of the National Computer Conference*, pages 589–596, 1975.

[3] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh – a generic and efficient polygon mesh data structure. In *OpenSG Symposium 2002*, 2002.

[4] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges — A scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–12, 1998.

[5] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[6] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, Sept. 1978.

[7] X. Gu and S.-T. Yau. Global conformal surface parameterization. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 127–137, 2003.

[8] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 221–234, 1983.

[9] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 99–108, 1996.

[10] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 13(1):65–90, May 1999.

[11] L. Kobbelt. $\sqrt{3}$ subdivision. In *SIGGRAPH '00 Conference Proceedings*, pages 103–112, 2000.

[12] W. B. R. Lickorish. Simplicial moves on complexes and manifolds. In *Proceedings of the Kirbyfest*, volume 2, pages 299–320, 1999.

[13] C. T. Loop. Smooth subdivision surfaces based on triangles, 1987. Master's Thesis, Department of Mathematics, University of Utah.

[14] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, New York, NY, 1988.

[15] K. Pulli and M. Segal. Fast rendering of subdivision surfaces. In *Proceedings of the EUROGRAPHICS Workshop on Rendering Techniques*, pages 61–70, 1996.

[16] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[17] L.-J. Shiue, P. Alliez, R. Ursu, and L. Kettner. A tutorial on cgal polyhedron for subdivision algorithms. In *2nd CGAL User Workshop*, 2004. http://www.cgal.org/Tutorials/Polyhedron/.

[18] L.-J. Shiue, V. Goel, and J. Peters. Mesh mutation in programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 15–24, 2003.

[19] L.-J. Shiue and J. Peters. A mesh refinement library based on generic design. In *Proceedings of the 43rd Annual ACM Southeast Conference*, pages 1–104–1–108, 2005.

[20] A. Sovakar and L. Kobbelt. API design for adaptive subdivision schemes. *Computers & Graphics*, 28(1):67–72, Feb 2004.

[21] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann Publishers, 2002.

[22] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, Jan. 1985.

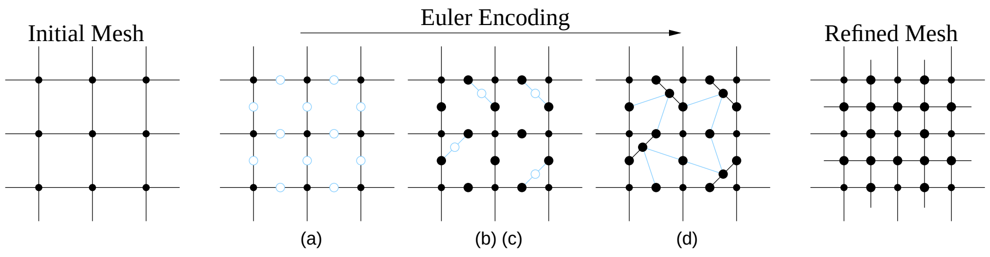


Figure 7: An Euler encoding of the PQQ refinement (see Listing 1). Light blue indicates the insertion of the new vertex and edge.

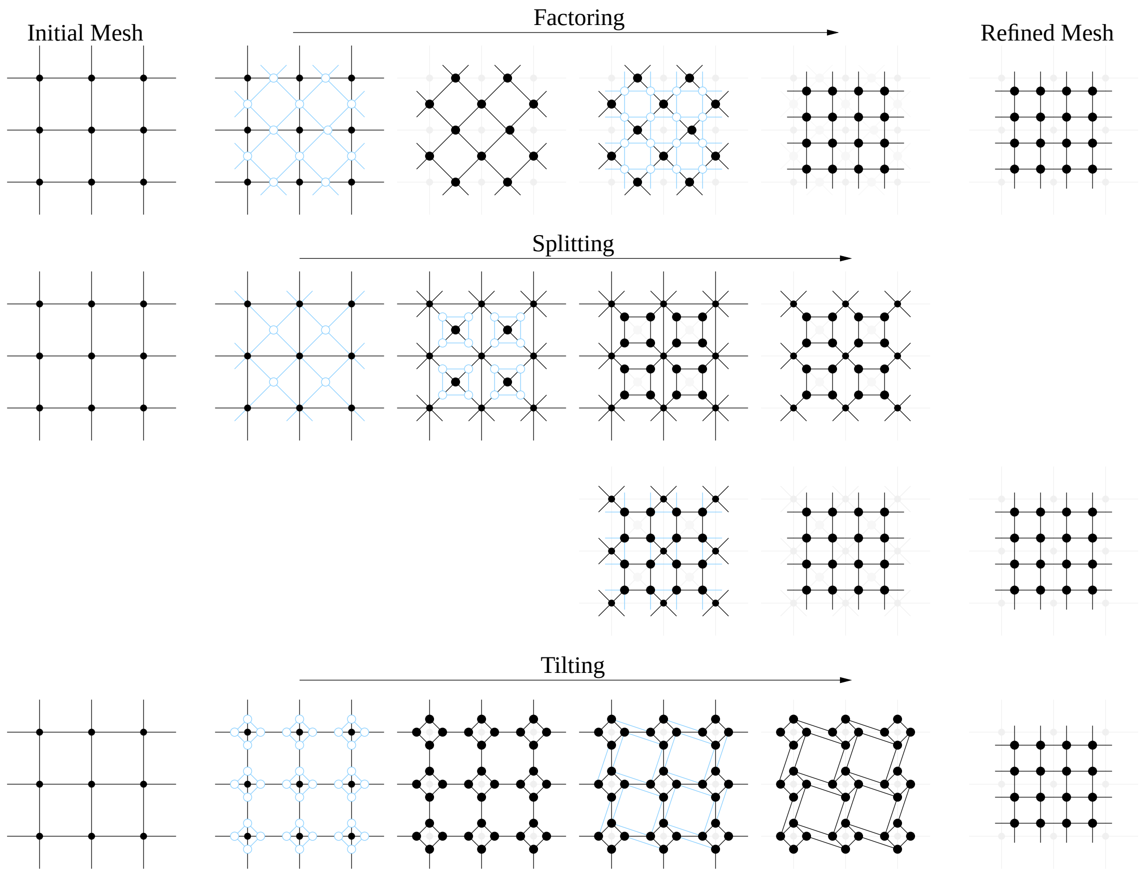


Figure 8: Three alternative Euler encodings of the DQQ refinement (see Listing 2 and Listing 3). Light blue indicates the insertions of the vertex and the edge.

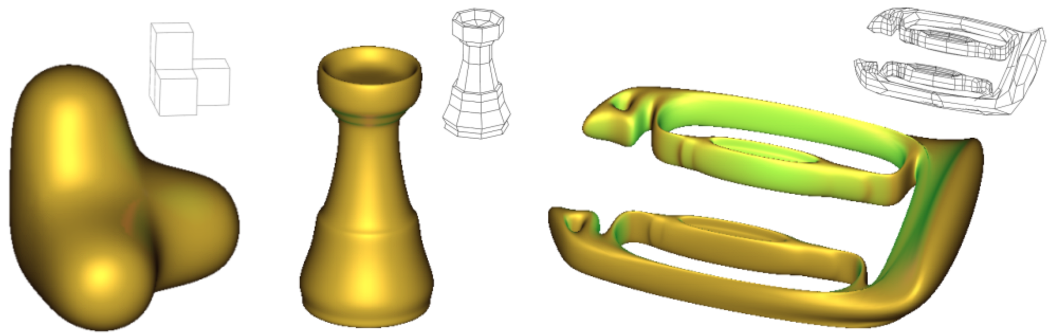


Figure 9: Test models of the Euler encoding: tripod, rook, and gripper rendered with Catmull-Clark subdivision.