

Progressive Combined B-reps Multi-Resolution Meshes for Interactive Real-Time Shape Design

Sven Havemann

CGV, TU Graz

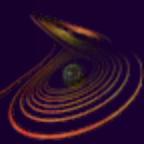
Austria

Dieter Fellner

CGV, TU Graz

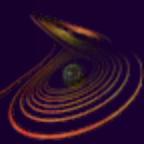
GRIS, TU Darmstadt

Fraunhofer IGD



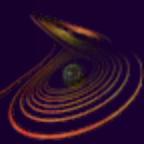
Motivation

- Quest for a suitable **data structure** for **interactive shape design** that
 - ❖ Can be *rendered efficiently*
 - Adaptive tessellation → Multi-resolution surfaces
 - ❖ Has an *adequate number of DOFs*
 - Comfort vs. Control: Metaballs vs. Triangle meshes
 - ❖ Can be *updated at real-time*
 - No pre-processing
- Problem: Updates invalidate cached data
- **Moore's Law:** Update-friendly data structures become increasingly important!

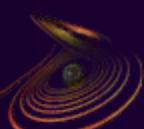
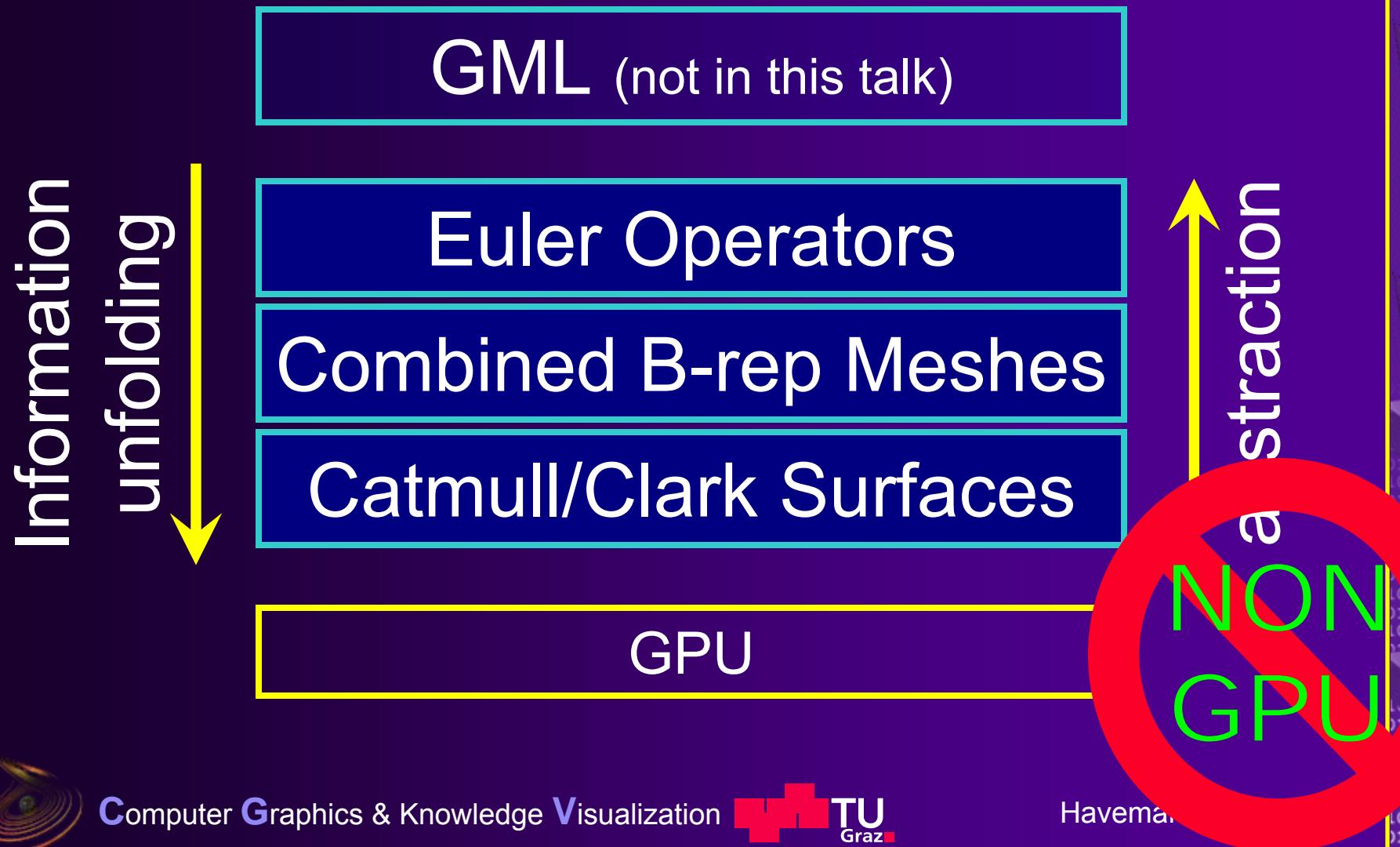


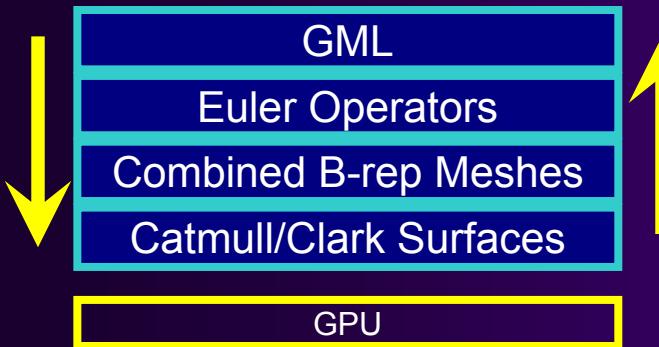
Related Work

- Surprisingly few work on **shape data structures** especially for **interactive design**
- Much work on **adaptive** rendering schemes that require substantial **pre-processing**
 - ❖ Simplification → Progressive Meshes
- Most papers on **interactive shape design** focus on modeling, not on data structures
 - ❖ Modeling w/ triangle meshes, implicit surfaces, volumetric simplicial complexes, subdivision solid, point clouds, deforming NURBS, ...
- Main problem: Maintain consistent multi-tess



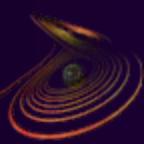
We propose a Layered Software Architecture





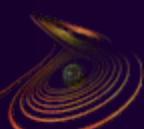
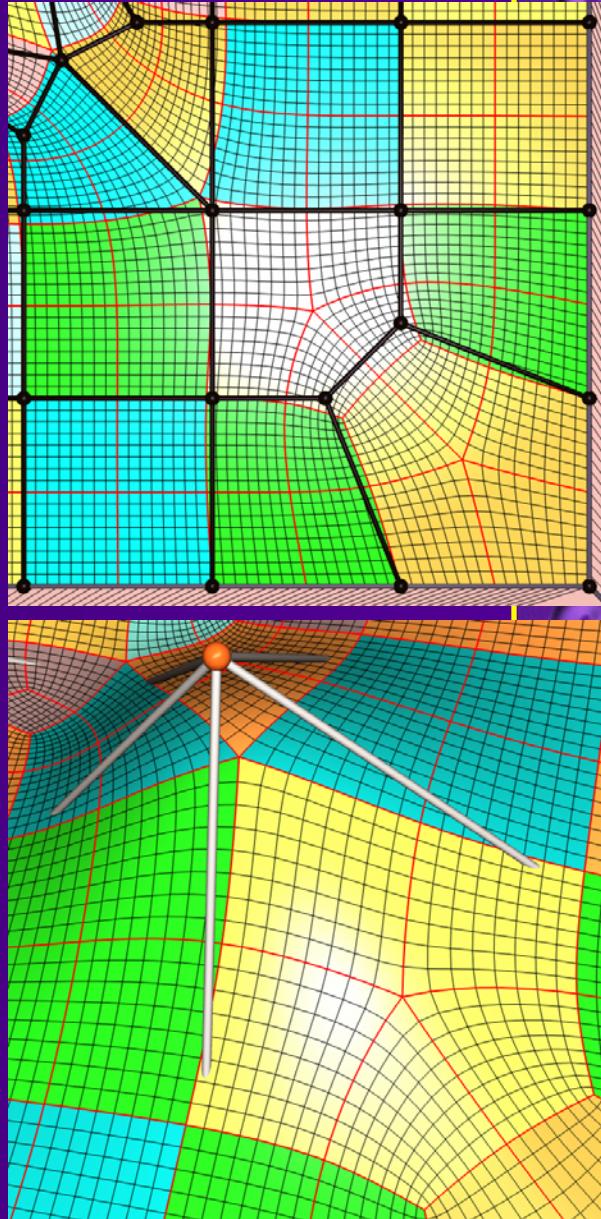
Layer 1: Catmull/Clark Surfaces

Reduction of complexity
for smooth surfaces of more than
two orders of magnitude



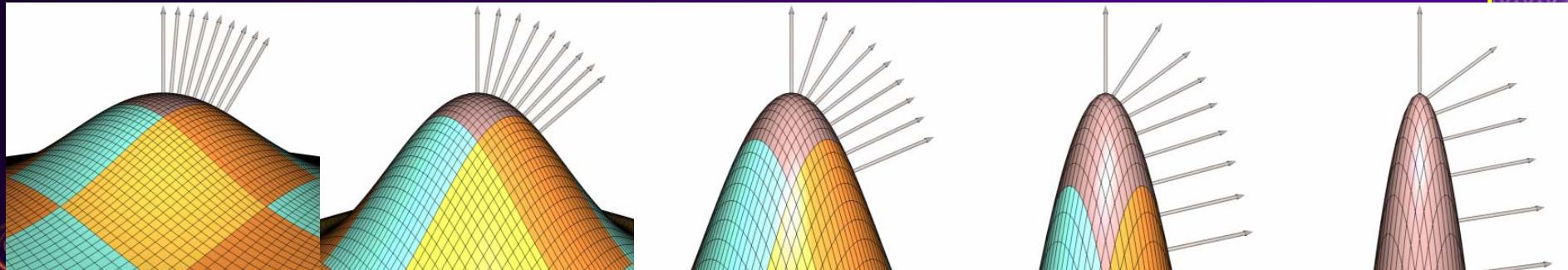
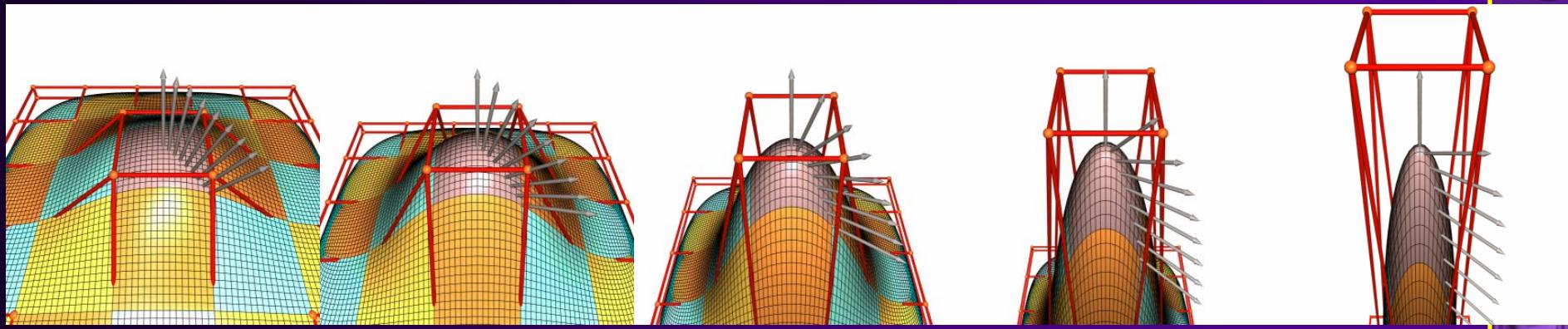
Optimized Recursive Subdivision

- Catmull/Clark surfaces are C^2 almost everywhere
 - ❖ Except at irregular vertices (valence $\neq 4$)
- Max subdivision level: $1+3 = 4$
 - ❖ Good arguments for that 
- Control face with n vertices \rightarrow n quad patches, 9×9 vertices
- Quad face of control mesh \rightarrow $16\times 16 = 256$ tess quads



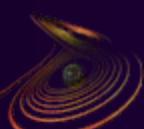
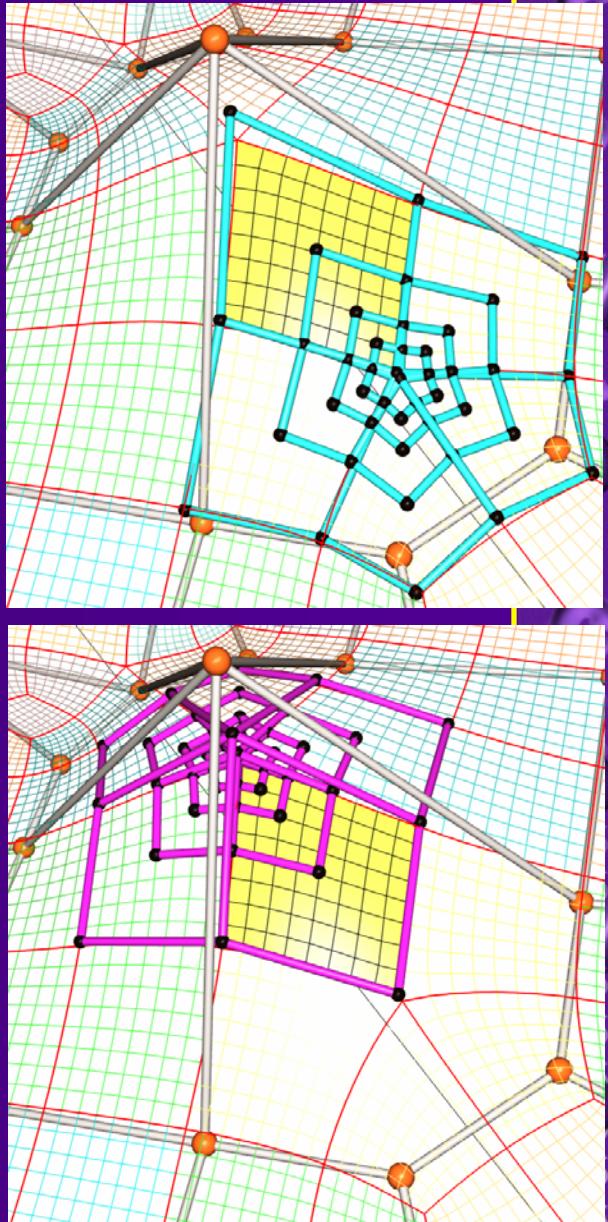
Max. Subdivision Level 4

- Angle between tess quads depends on
 - ❖ Angle between control mesh faces
 - ❖ Size of control mesh faces
- Worst case test: Small face, large angle

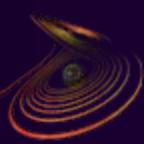
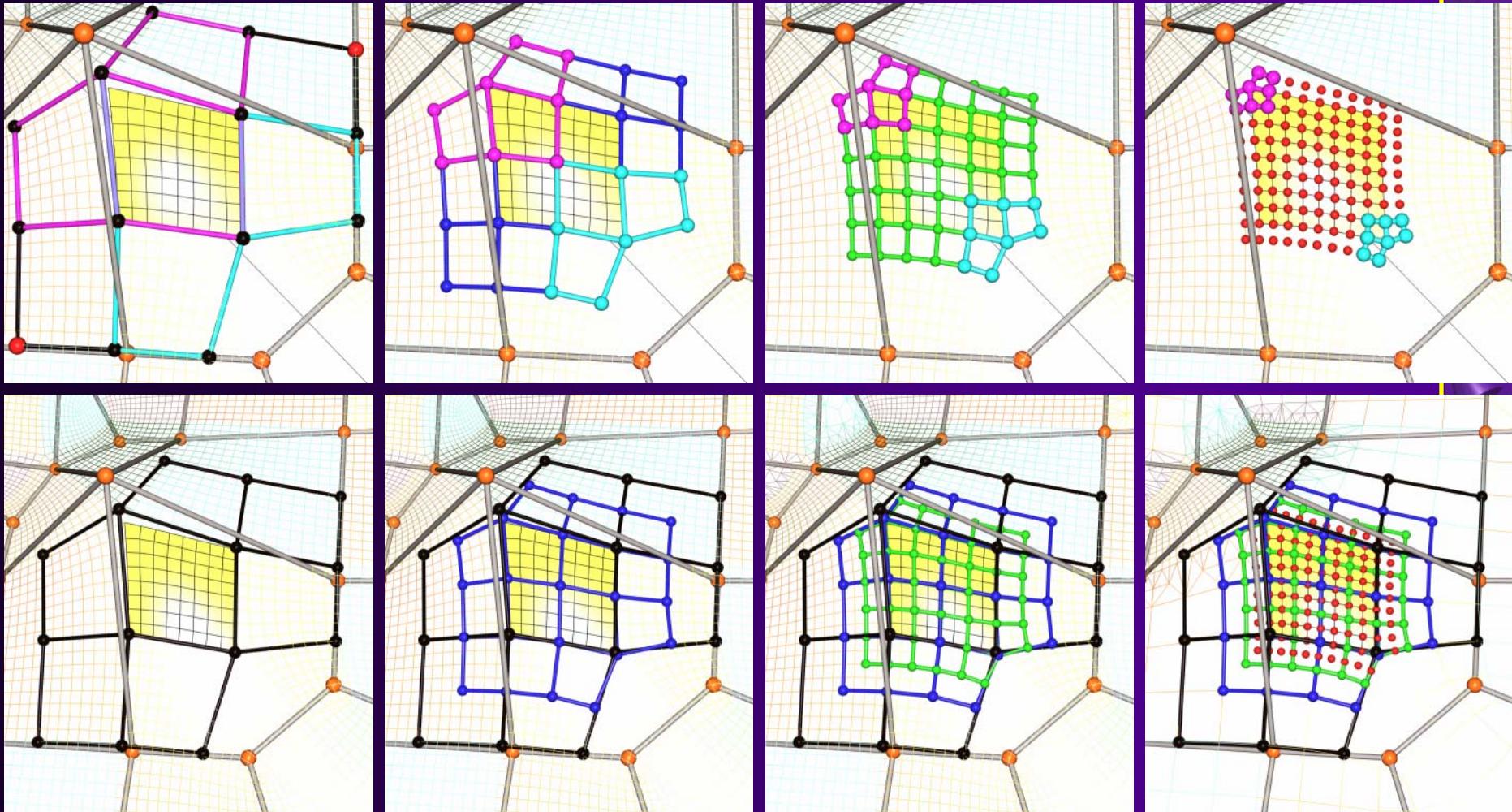


Vertex and Face Rings

- Separate *regular* from *irregular*
 - ❖ Irregular vertices: Valence $\neq 4$
 - ❖ Irregular faces: Degree $\neq 4$
 - ❖ Regular: 9×9 patches
 - Highly optimize regular tessellation!
- Ring: Center + 1-neighbour.
 - ❖ Enough data for subdivision
- Catmull/Clark Patch:
 - ❖ Fixed-size data structure
 - ❖ 81 vertices, 81 normals
 - ❖ $\text{Sizeof(Patch)} = 2016$ bytes

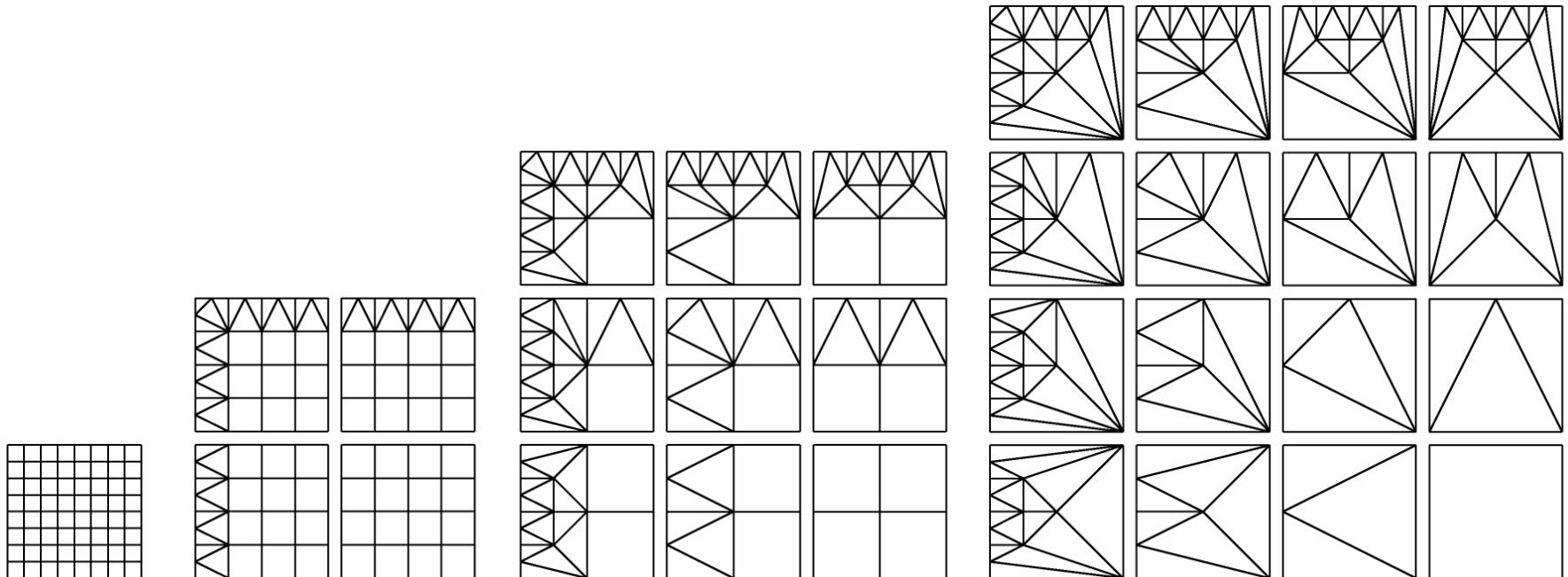


Fast Recursive Subdivision

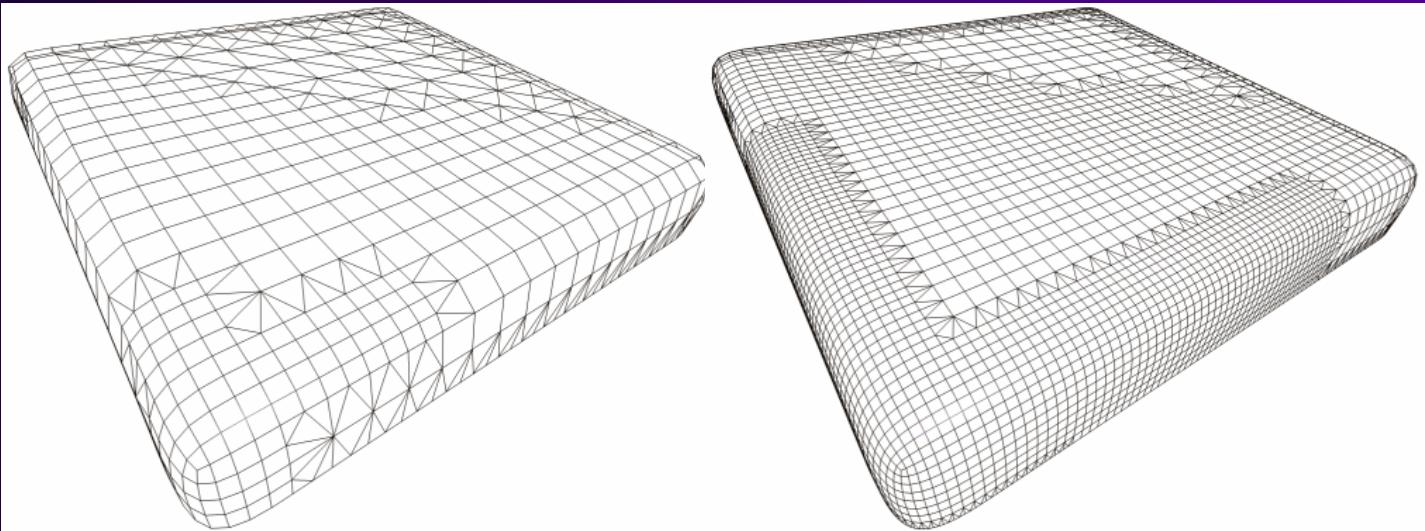


Adaptive Tesselation – on the fly!

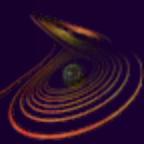
- Table of pre-defined strip indices
- Refinement towards neighbour face
- Switch resolution without any further computations – once the patches are filled



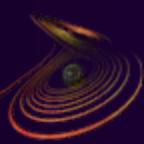
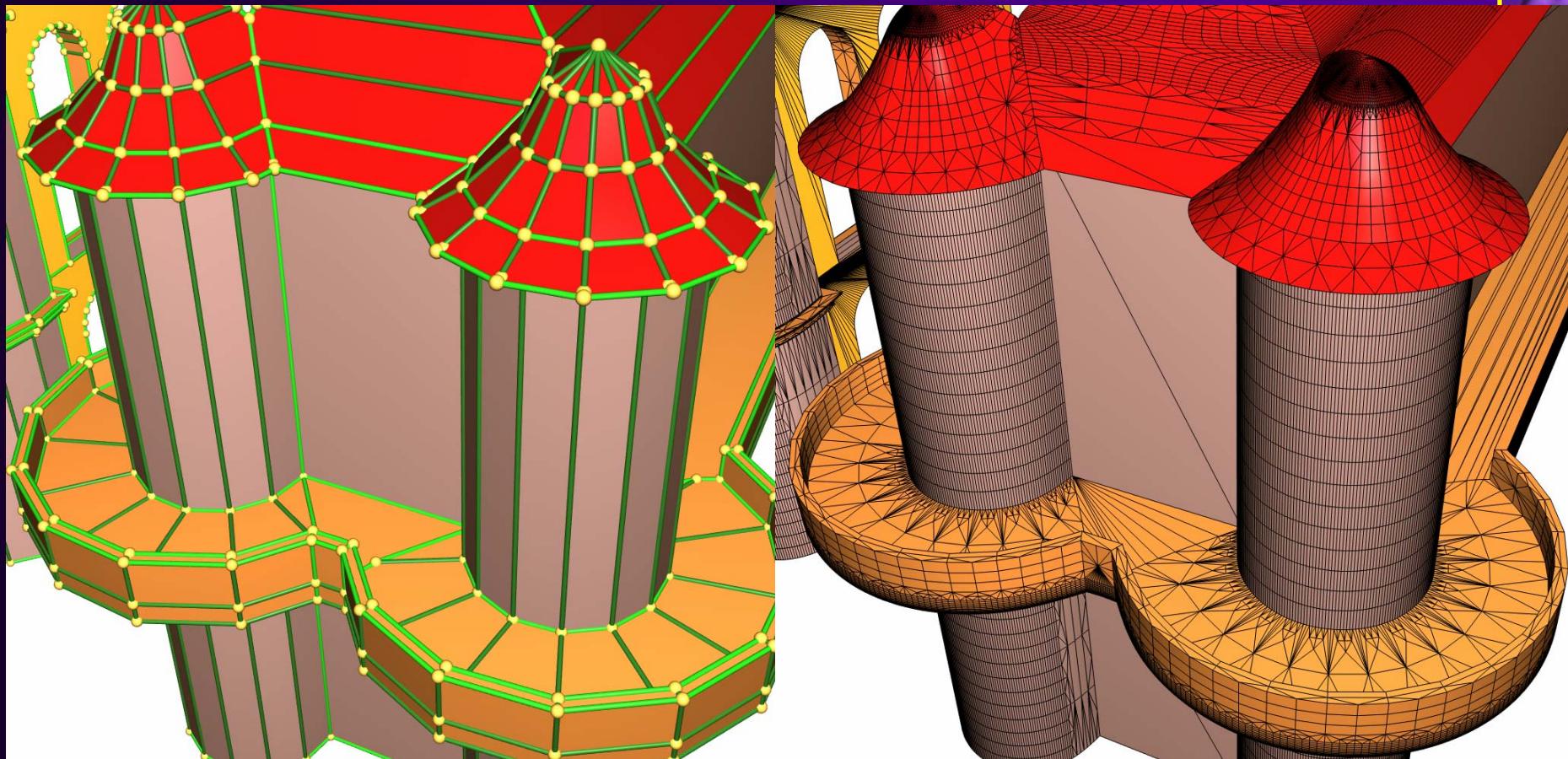
Adaptive Real-Time Tesselation

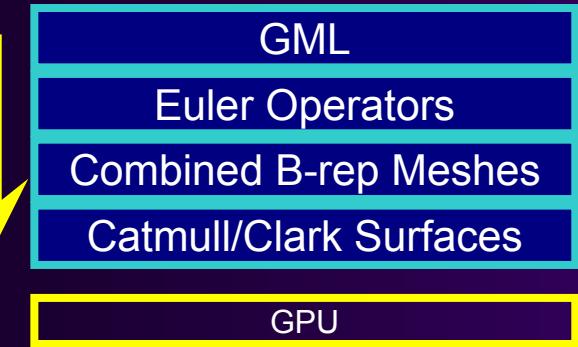


Havemann/Fellner,
Interactive Rendering of
Catmull/Clark Surfaces with
Crease Edges,
The Visual Computer (18)
2002



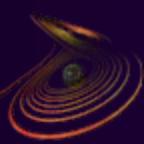
Adaptive Real-Time Tesselation



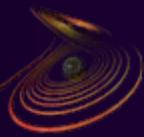
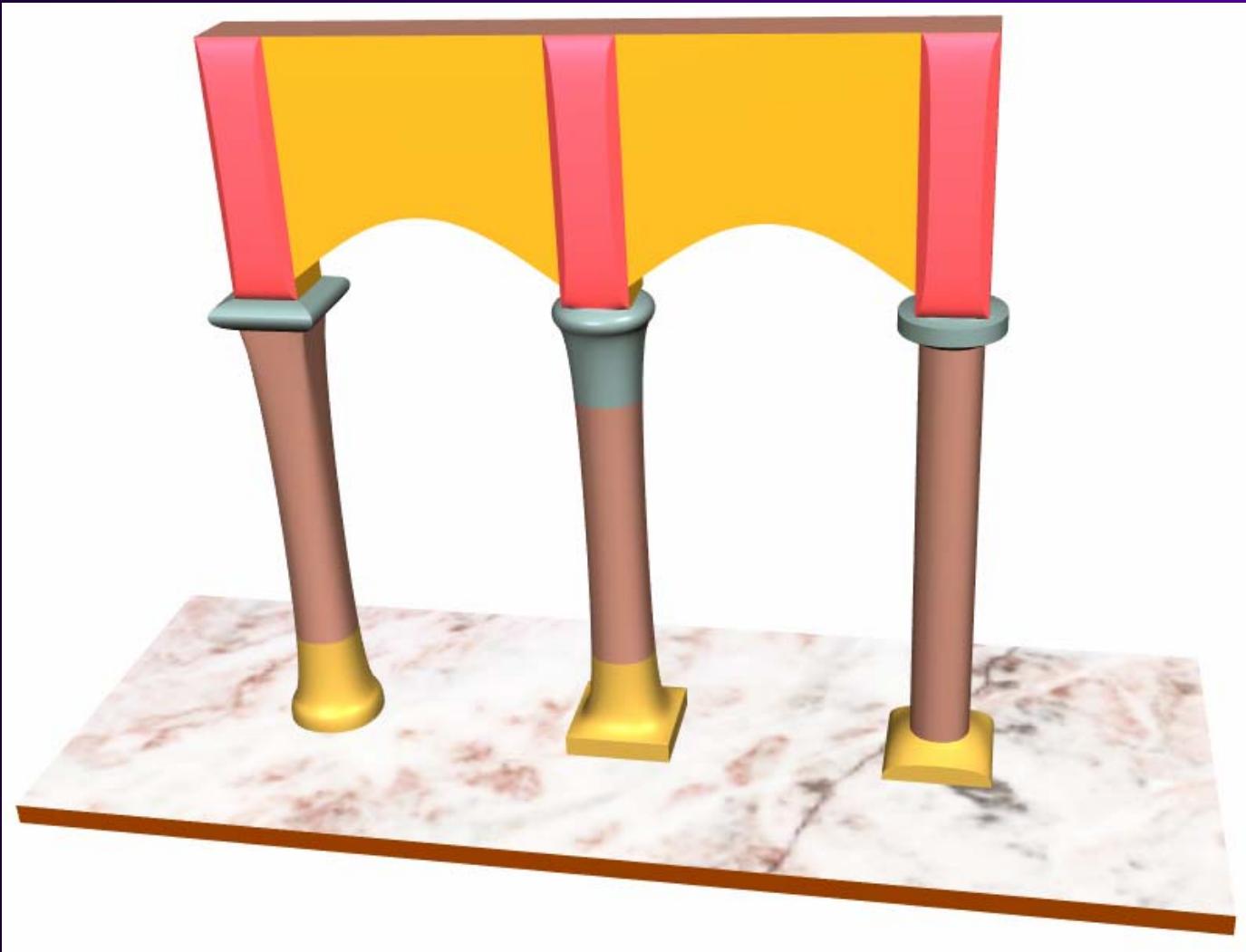


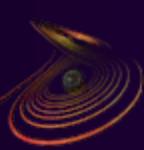
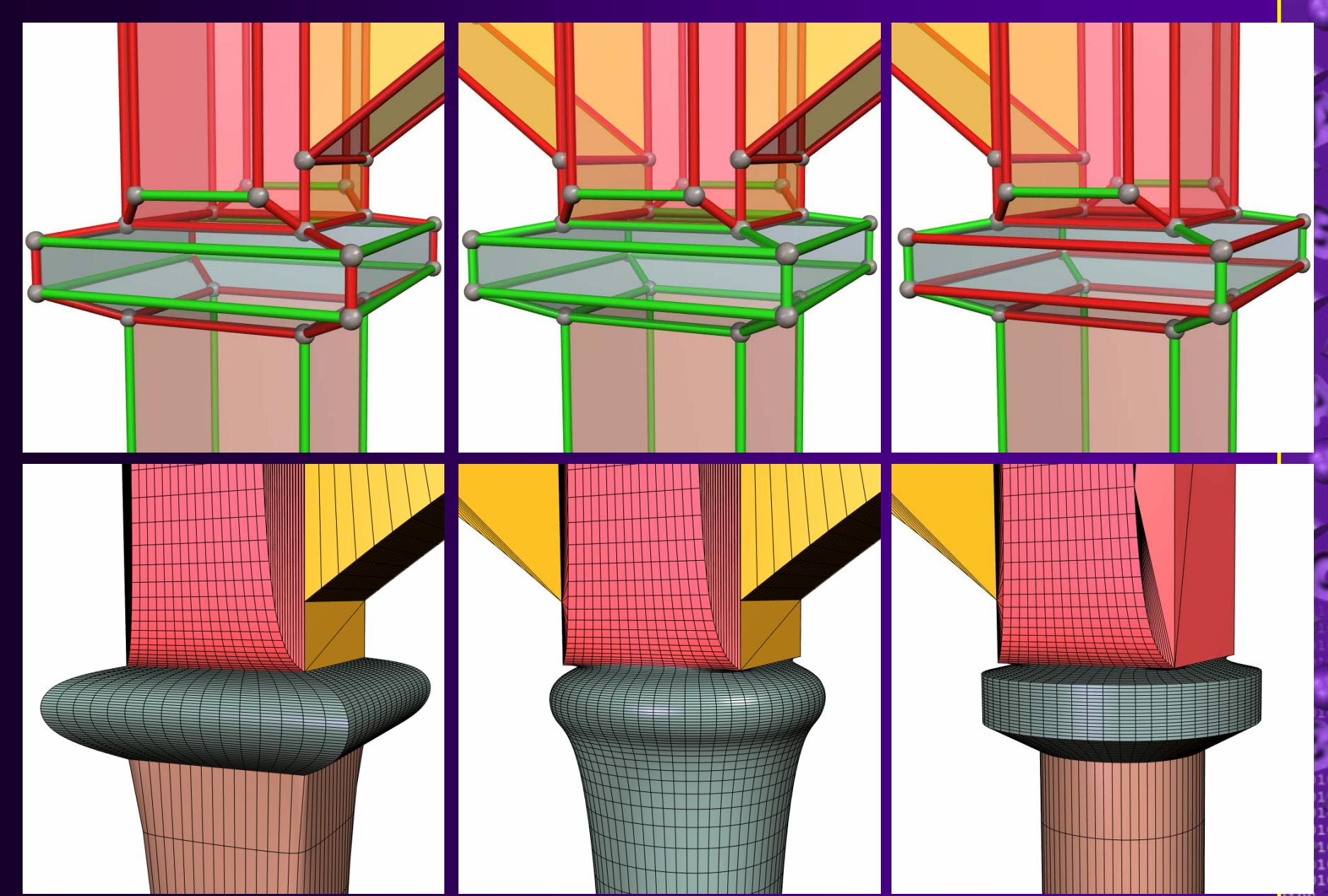
Layer 2: Combined B-Reps

Combination of
polygonal models with
freeform models



Layer 2: Combined B-Reps





Classification Rules

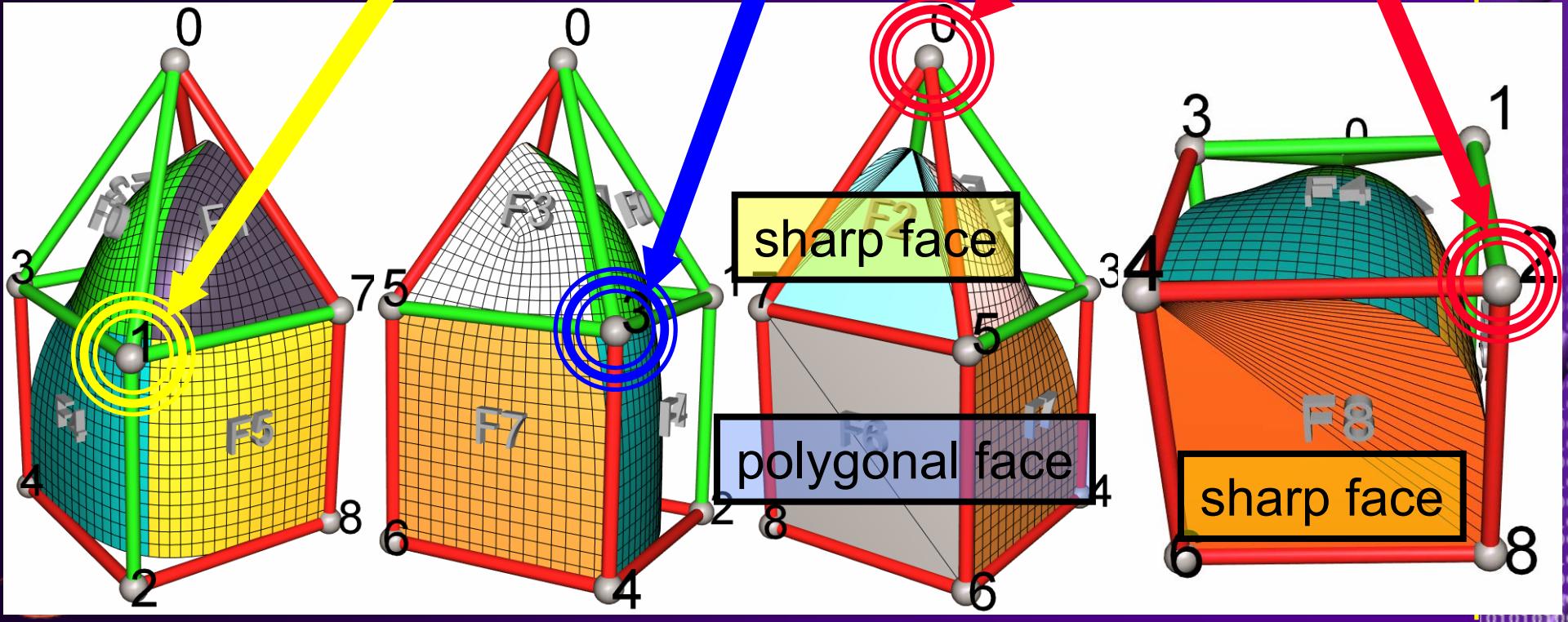
- **Smooth vertex** 0 sharp edges
 - ❖ control point of the freeform surface
- **Dart vertex** 1 sharp edge
 - ❖ endpoint of a crease curve
- **Crease vertex** 2 sharp edges
 - ❖ control point of a crease curve
- **Corner vertex** ≥ 3 sharp edges
 - ❖ vertex remains fix across all refinement levels

- **Smooth face** ≥ 1 smooth edge, no rings
- **Sharp face** all edges are sharp,
at least one crease vertex
- **Polygonal face** all edges are sharp
and all vertices are corners



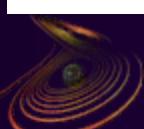
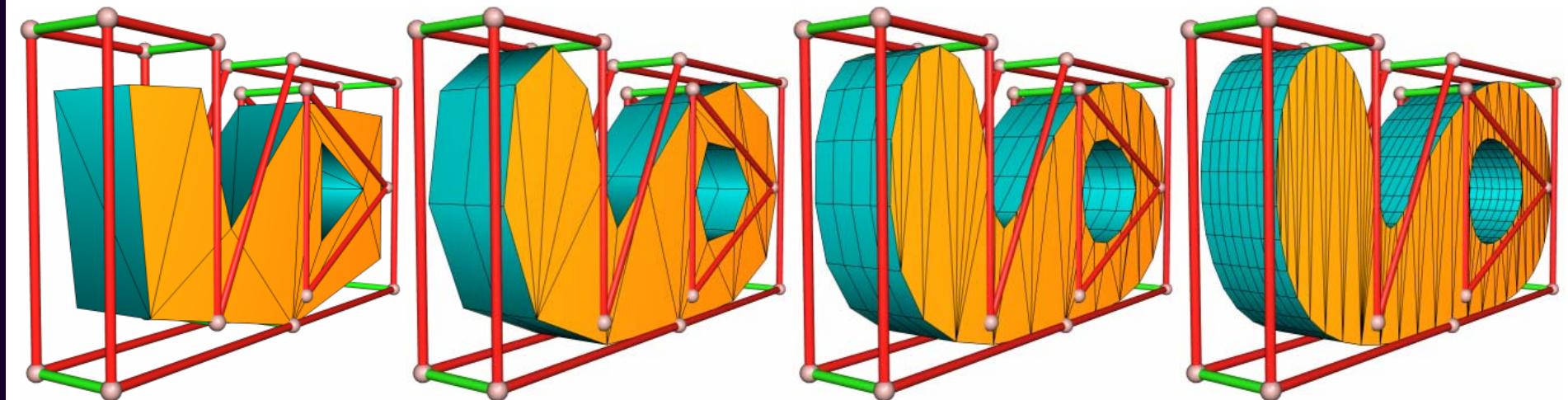
Classification Example

- Vertices: 1 smooth
- 3 dart
- 0 2 crease
- Others are corner



Tesselation of Sharp Faces

- Four triangulations, one for each depth 1-4
- Essentially just doubles space: $n+n/2+n/4\dots$
 - ❖ Polygon with n vertices has $n-2$ triangles
- Depth of sharp face = max neighbour depth
 - ❖ Smooth faces can refine towards neighbour!



```

template<class Trait>
struct BRepMesh
{
    typedef BRepMesh< Trait> Mesh ;
    typedef typename Trait::V V;
    typedef typename Trait::E E;
    typedef typename Trait::F F;

    struct Vertex { . . . see code to the right }
    struct Edge { . . . see code to the right }
    struct Face { . . . see code to the right }

    BRepMesh () {}
    ~BRepMesh () {}

    void clear ();
    bool reserve ( int kv , int ke , int kf );
    void purge ( int* cv , int* ce , int* cf );
    bool relocate ();

    Skipvector<Face> faces ;
    Skipvector<Edge> edges ;
    Skipvector<Vertex> vertices ;
};


```

```

struct Vertex : public V
{
    Vertex () {}
    Vertex ( const V& data ) : V( data ) {}
    int status ;
    Edge* oneEdge ;
};

```

```

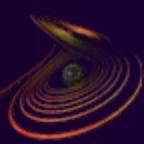
struct Edge : public E
{
    Edge () {}
    Edge( const E& data ) : E( data ) {}
    int status ;
    Vertex * vertex ;
    Edge* next ;
    Face* face ;
};

```

```

struct Face : public F
{
    Face () {}
    Face ( const F& data ) : F( data ) {}
    int status ;
    Edge* oneEdge ;
    Face* nextring ;
    Face* baseface ;
};

```



```

struct BRepCombinedTrait
{
    struct V {
        typedef enum { CornerVertex ,
                        CreaseVertex ,
                        DartVertex ,
                        SmoothVertex
                    } Type ;
        Vec3f position ; // INPUT DATA
        Type type ;
        int ringID ;
    } ;

    struct E {
        bool sharp ; // INPUT DATA
        int patchID ; // Catmull/Clark
        int sourceID ; // pcB-rep ID
    } ;

    struct F {
        (..... See code to the right)
    } ;
};

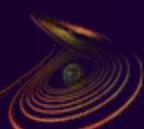
```

```

struct F {
    typedef enum { HollowFace ,
                    SmoothFace ,
                    SharpFace ,
                    PolygonalFace
                } Type ;
    int materialID ; // INPUT DATA
    int depth ; // PER FRAME
    Type type ;
    int ringID ;

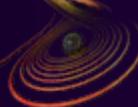
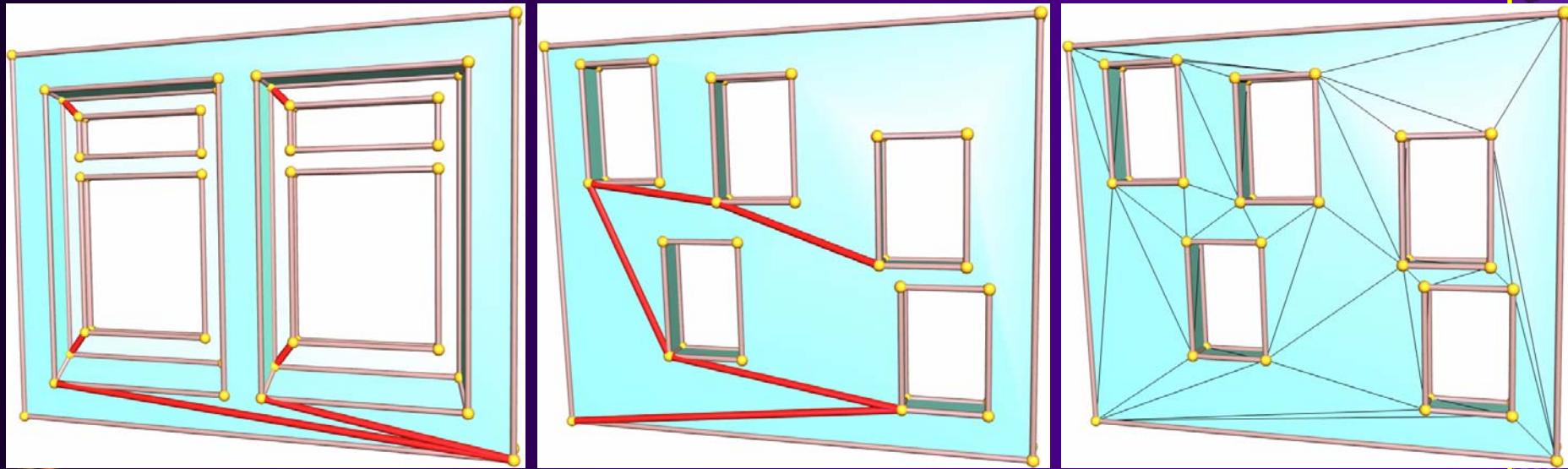
    int triChunk ;
    int sharpTriChunk [ 5 ] ;
    int sharpPtChunk ;
    Vec3f normal ;
    float normalDist ;
    float normalCone ;
    float sphereRad ;
    Vec3f sphereMid ;
};

```



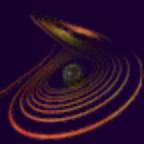
Faces with Rings

- Common feature in CAD software:
Faces can have multiple boundaries
 - ❖ One distinct outer boundary (CCW)
 - ❖ Multiple inner boundaries (CW)
- Advantage: No artifact edges to break up

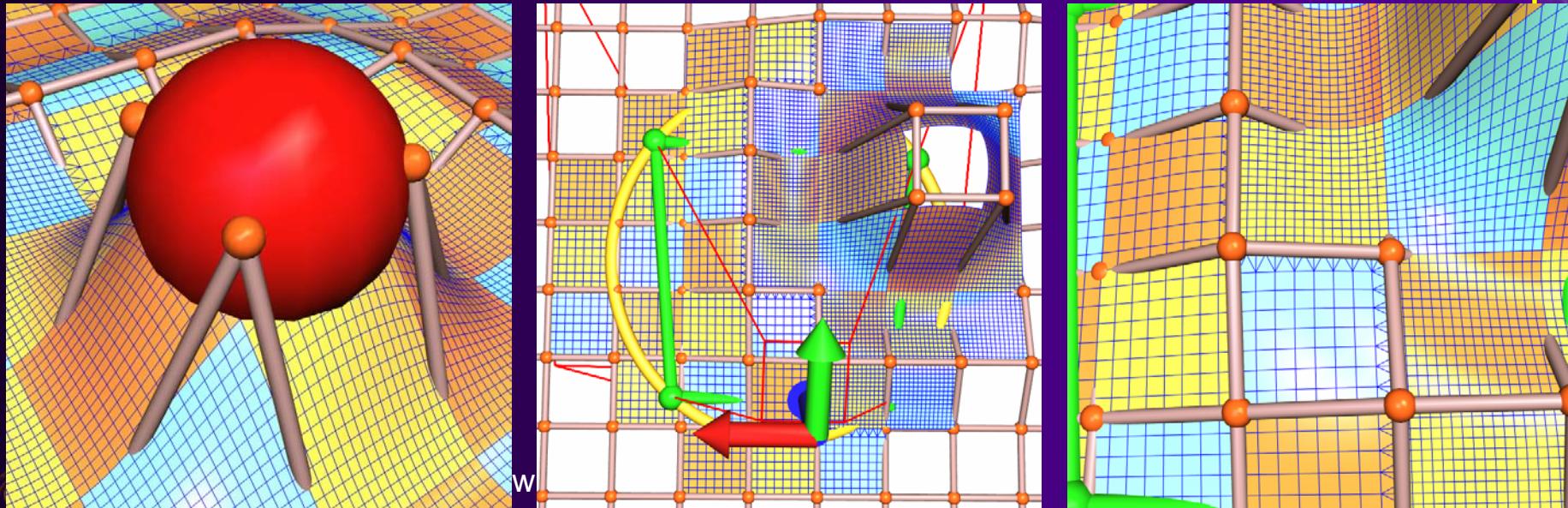
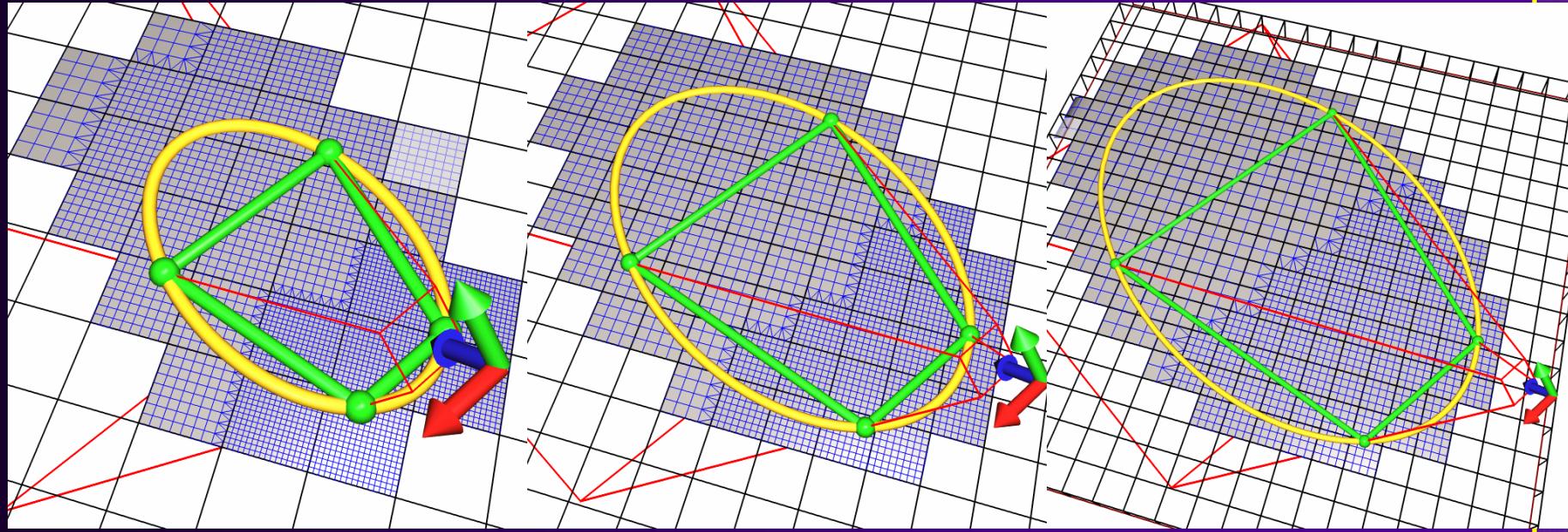


Skipvectors

- How to store highly dynamic data?
 - ❖ Doubly linked lists: `std::list<Item>`
 - Memory fragmentation, performance drops!
 - ❖ Arrays: `std::vector<item>`
 - Insertion/Deletion are $O(n)$!
- Solution: **Arrays with holes**
 - ❖ Bool `Item::isActive()`
 - ❖ Skip over inactive items
- Issue: **Array relocation when size doubles**
 - ❖ Pointers to array elements must be updated!

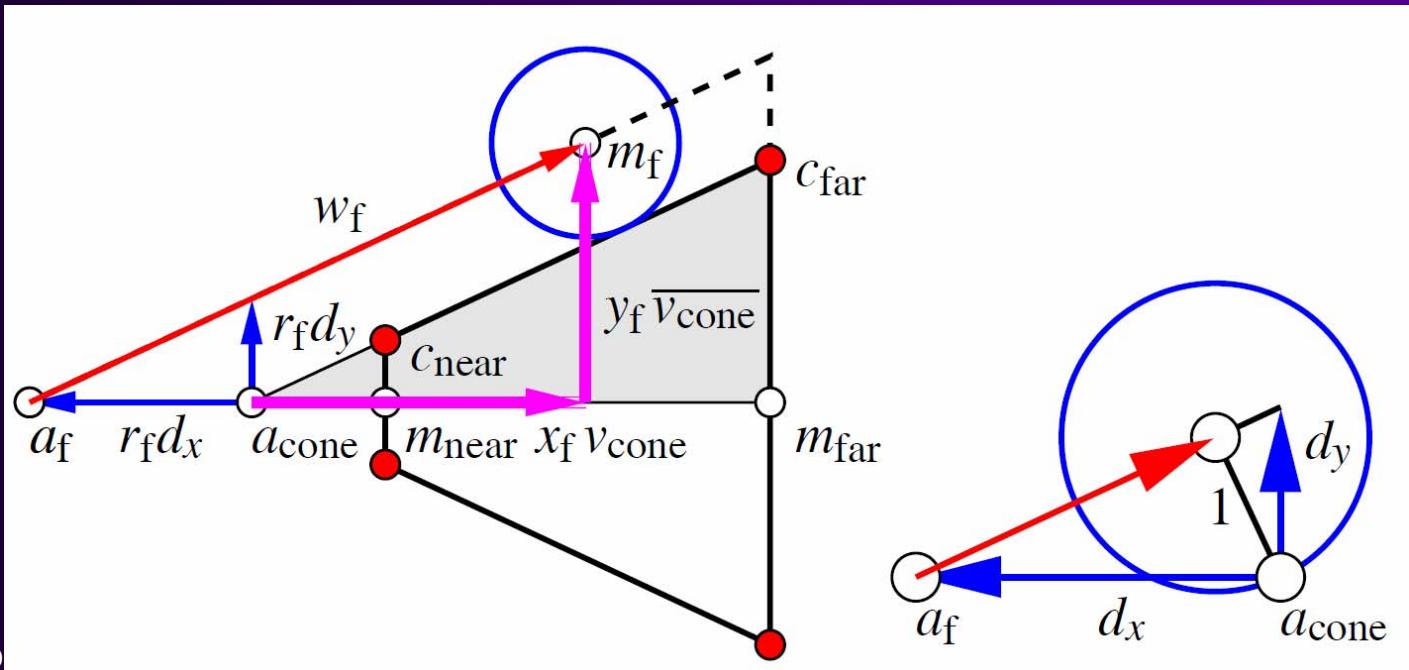


View Cone Culling



View Cone Culling

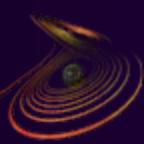
- Reduce sphere-in-cone test of bounding sphere to point-in-cone test
- Translation of the **apex** of the view cone
- Bounding sphere visible iff $y_f^2 / (x_f + r_f d_x)^2 < s_{\text{cone}}^2$ with $s_{\text{cone}} = |m_{\text{far}} - c_{\text{far}}| / |m_{\text{far}} - a_{\text{cone}}|$



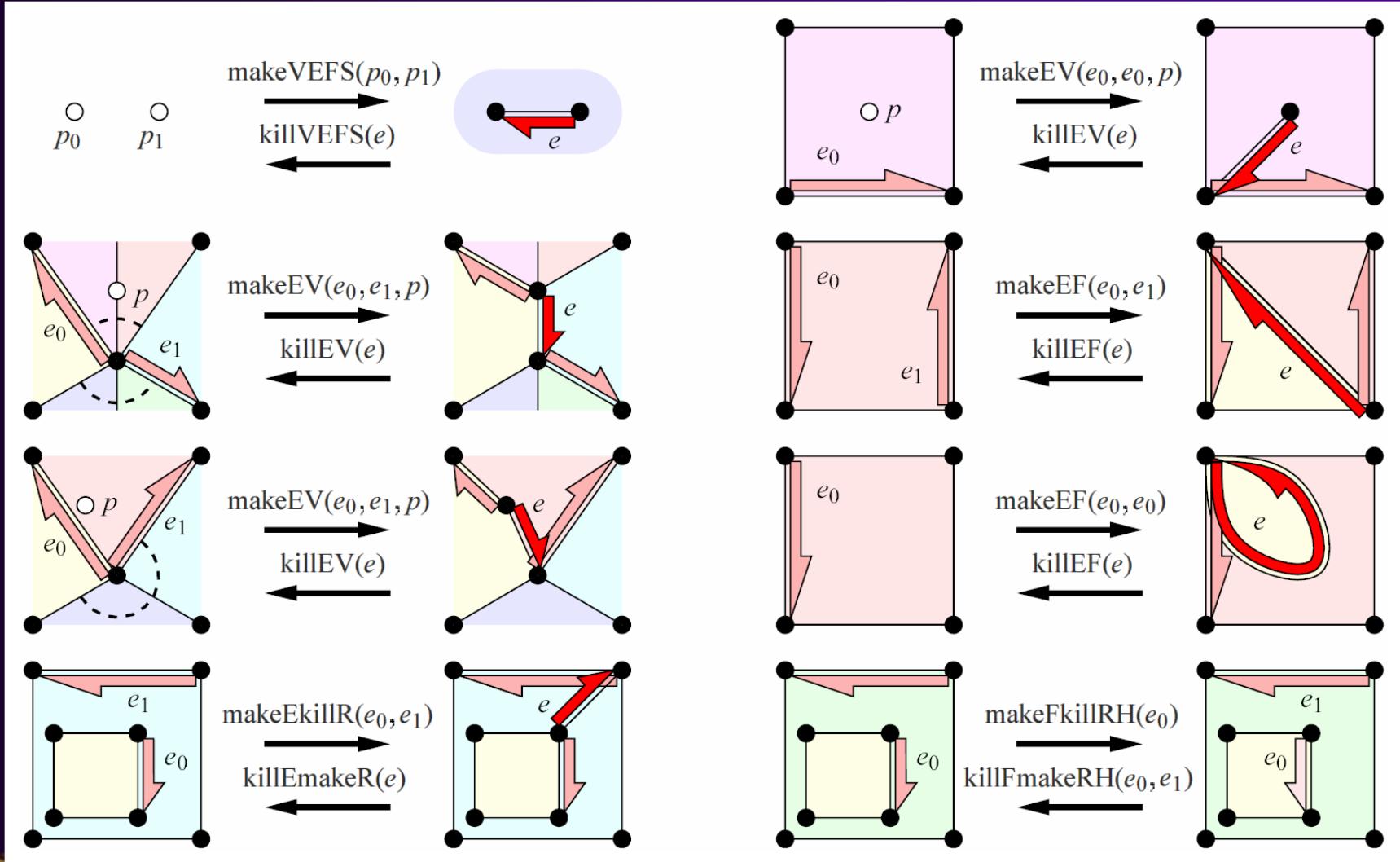


Layer 3: Euler Operators

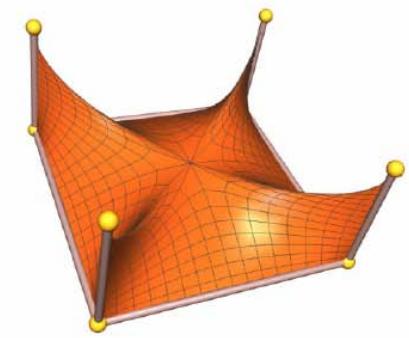
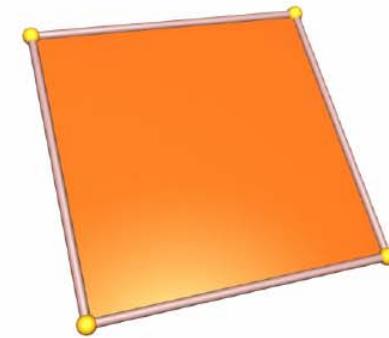
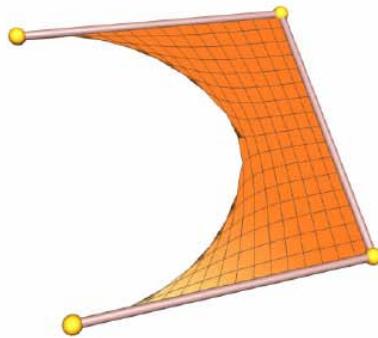
Paradigm Change
from Objects to Operations
for polygonal Meshes



5 Operators and 5 Inverse Ops



Example: Modeling with Euler Ops

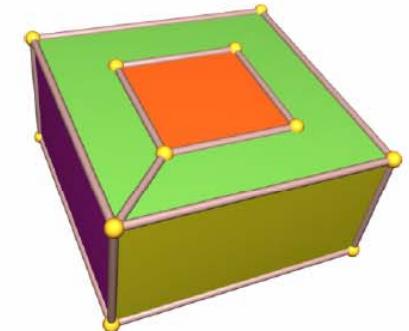
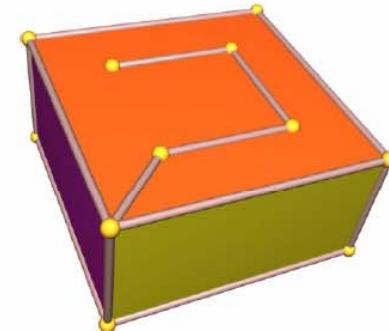
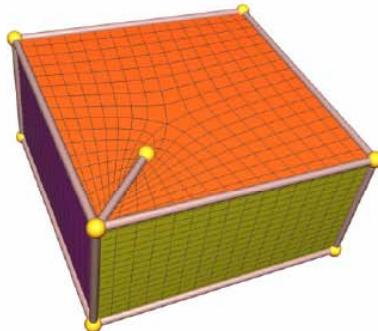
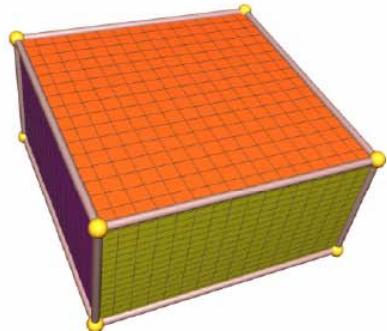


1: makeVFS

2: $3 \times$ makeEV (a,b,b)

3: makeEF (c)

4: $4 \times$ makeEV (b)

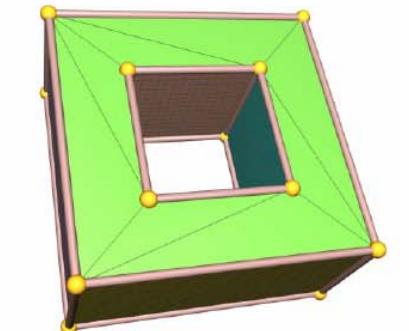
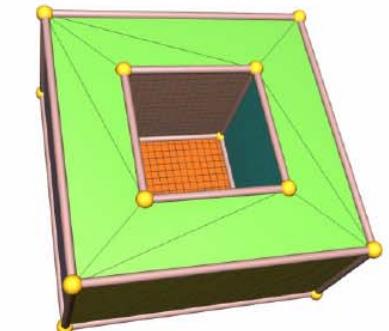
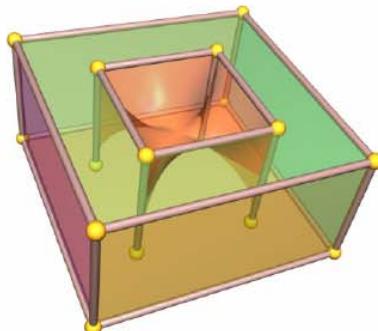
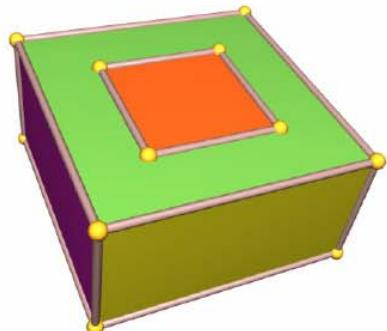


5: $4 \times$ makeEF (c)

6: makeEV (b)

7: $3 \times$ makeEV (b)

8: makeEF (c)



9: killEmakeR (c)

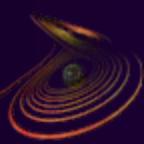
10: $4 \times$ makeEV (b)

11: $4 \times$ makeEF (c)

12: killFmakeRH

Progressive Combined B-Reps

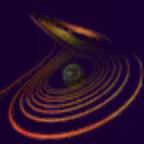
- Mesh access exclusively by Half-Edges
- Mesh manipulation exclusively through Euler operators [Mäntylä 88]
 - ❖ Complete + Sufficient on { manifold meshes }
 - Complete: No invalid mesh can be created
 - Sufficient: All valid meshes can be created
 - ❖ **Invertible:** Complete undo/redo history
 - ❖ Additional operators: moveV, sharpE
- Progressive Combined B-reps:
Store sequence of modeling operators



Subtlety: Operator Sequences

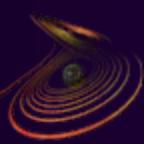
operation	edges	attributes	#
$e = \text{makeVEFS}(p_0, p_1, m, s)$ killVEFS(e)	e e	p_0, p_1, m, s $e \rightarrow \text{vertex} \rightarrow p, e \rightarrow \text{face} \rightarrow \text{material},$ $e \rightarrow \text{mate} \rightarrow \text{vertex} \rightarrow p, e \rightarrow \text{sharp}$	5 5
$e = \text{makeEV}(e_0, e_1, p, s)$ killEV(e)	e, e_0, e_1 $e, e \rightarrow \text{faceCCW}, (e \rightarrow \text{vertexCW})^*$	p, s $e \rightarrow \text{vertex} \rightarrow p, e \rightarrow \text{sharp}$	5 5
$e = \text{makeEF}(e_0, e_1, s, m)$ killEF(e)	e, e_0, e_1 $e, (e \rightarrow \text{faceCCW})^*, e \rightarrow \text{vertexCW}$	s, m $e \rightarrow \text{sharp}, e \rightarrow \text{face} \rightarrow \text{material}$	5 5
$e = \text{makeEkillR}(e_0, e_1, s)$ killEmakeR(e)	e, e_0, e_1 $e, e \rightarrow \text{faceCCW}, e \rightarrow \text{vertexCW}$	s $e \rightarrow \text{sharp}$	4 4
makeFkillRH(e, m) killFmakeRH(e_0, e_1)	$e, e \rightarrow \text{face} \rightarrow \text{baseface} \rightarrow \text{oneEdge}$ e_0, e_1	m $e_0 \rightarrow \text{face} \rightarrow \text{material}$	3 3
moveV(e, p) sharpE(e, s) materialF(e, m)	e e e	$p, e \rightarrow \text{vertex} \rightarrow p$ $s, e \rightarrow \text{sharp}$ $m, e \rightarrow \text{face} \rightarrow \text{material}$	3 3 3

- Do not reference the edges by their array indices!
 - ❖ Redo might re-create edge with different array index
- Instead: Store id of **edge-creating Euler operation**



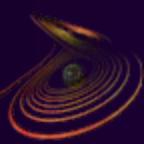
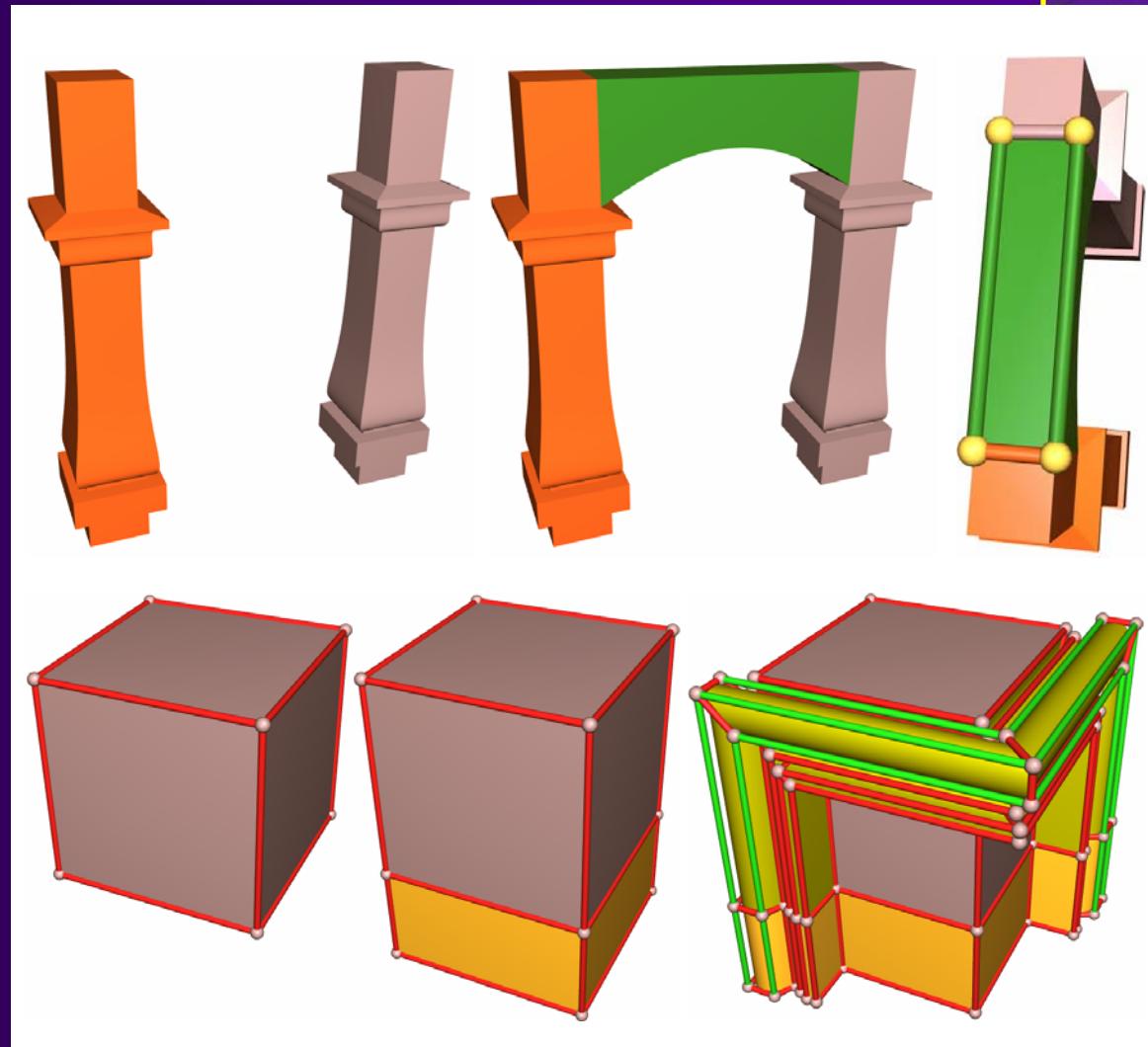
Undo: Sequence Inversion

- Inverse of operator sequence:
 - ❖ Inverse each operator and
 - ❖ Reverse the sequence
- Difference to progressive triangle meshes:
Also **destructive** operators can be used
- Sequence: $[\dots, op_i(\text{makeEV}), op_{i+1}(\text{killEV}), \dots]$
Inverse: $[\dots, inv_{i+1}(\text{makeEV}), inv_i(\text{killEV}), \dots]$
- Also note: Halfedges encode a direction!
 - ❖ Solution: Take 2i or 2i+1 as indices



Dependencies in Operator Sequence

- Important:
Undo/Redo can
be **out of order**
- Operator
dependency
DAG
- **Undo(op)**: first
undo *later* ops
- **Redo(op)**: first
redo *earlier* ops





Thank you for your attention!

Google Query: “havemann mesh”
www.generative-modeling.org

