WITH OR WITHOUT EDGES?

DATA STRUCTURE FOR GLOBAL MESH OPERATIONS

Gábor Fábián, Eötvös Loránd University, Budapest, Hungary

Motivation

>> Why we need new data structure?

>> Modeling

>> Frequent (local) manipulations.

>> No need for real-time operations.

>> Computer Graphics

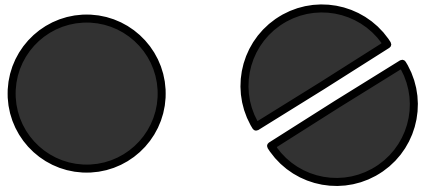
>> No manipulations.

>> Real-time rendering.

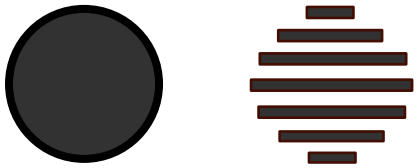
>> Physical simulation

>> Real-time global manipulations.

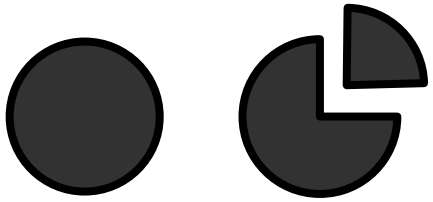
>> Real-time rendering.



Splitting / Cutting



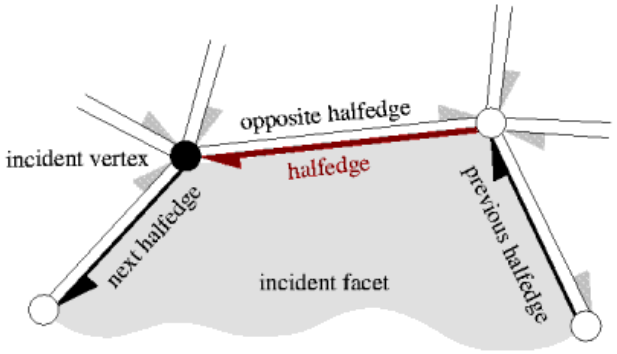
Slicing



Fracturing

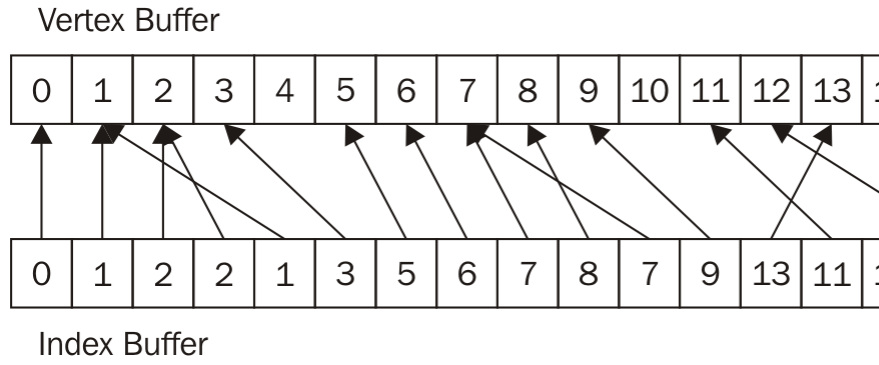
CAD vs CG

Forrás: https://doc.cgal.org/latest/HalfedgeDS/halfedge_small.png



CAD + CG?

Source: https://www.oreilly.com/api/v2/epubs/9781786468154/files/assets/image_09_01-1.png



Fast traversal	Fast traversal	Slow traversal
Slow render	Fast render	Fast render
High overhead	Medium overhead	No overhead
Explicit edges	Implicit edges?	No edges

Principles of design

Explicit use of vertex array & index array

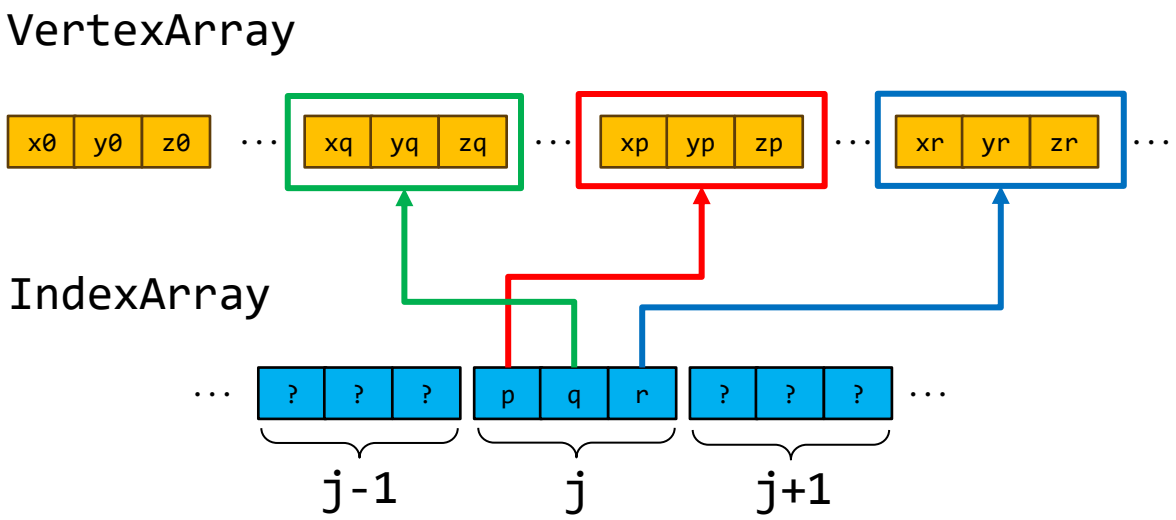
No edges

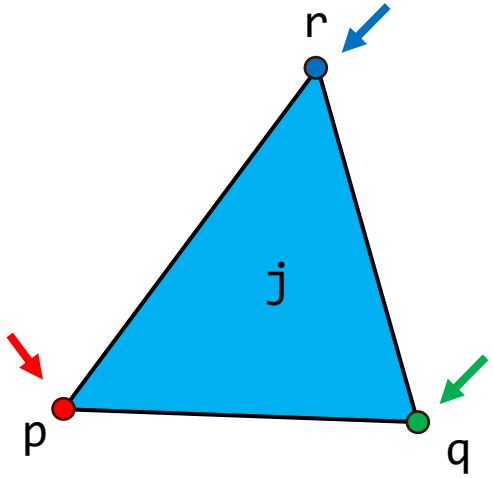
Fixed amount of data stored per vertex / triangle

Fast global operations

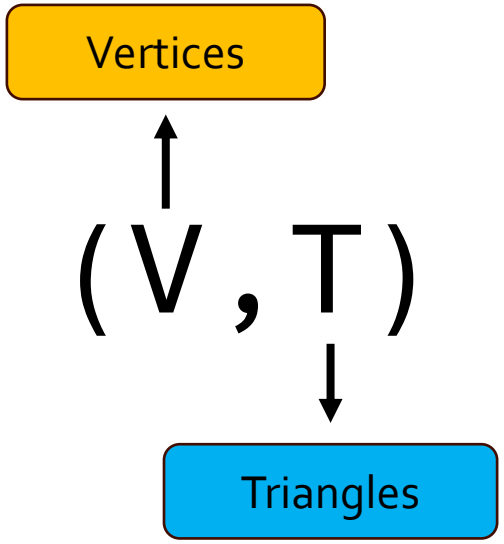
Triangle mesh

VertexArray & IndexArray



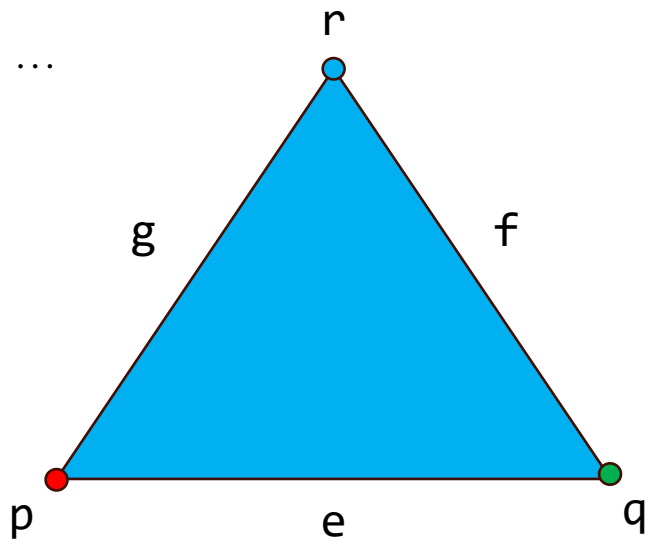


Designing data structure 1/3



Explicit edge representation

EdgeArray (?)



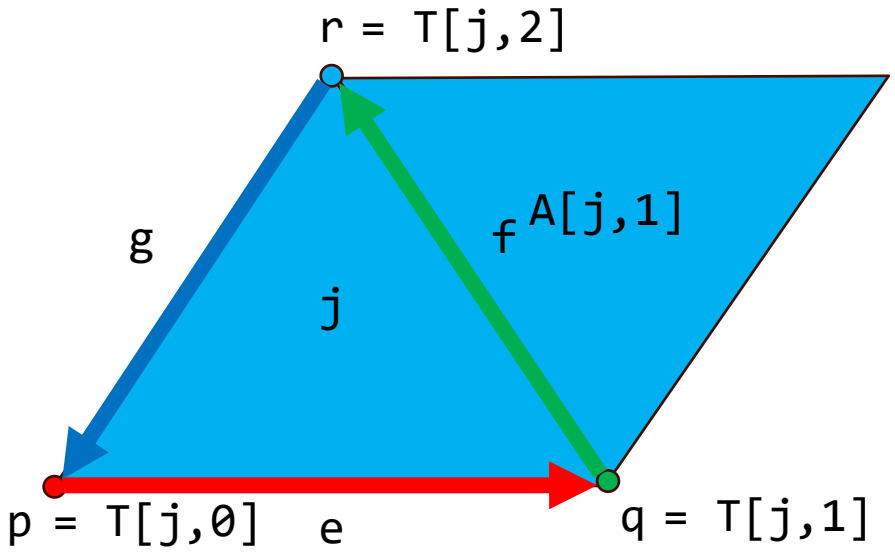
Edges without edges?

>> What is an edge?

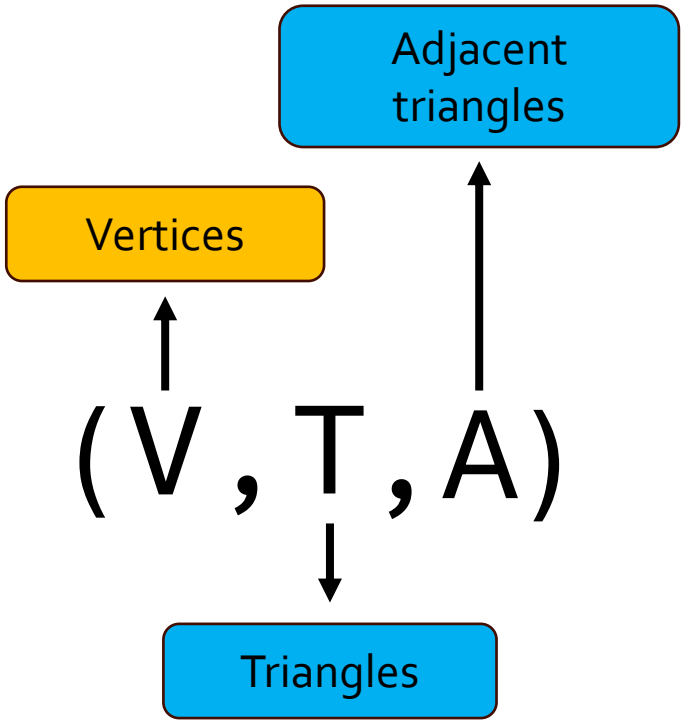
>> Edge of a triangle =

Triangle index + (starting) vertex index

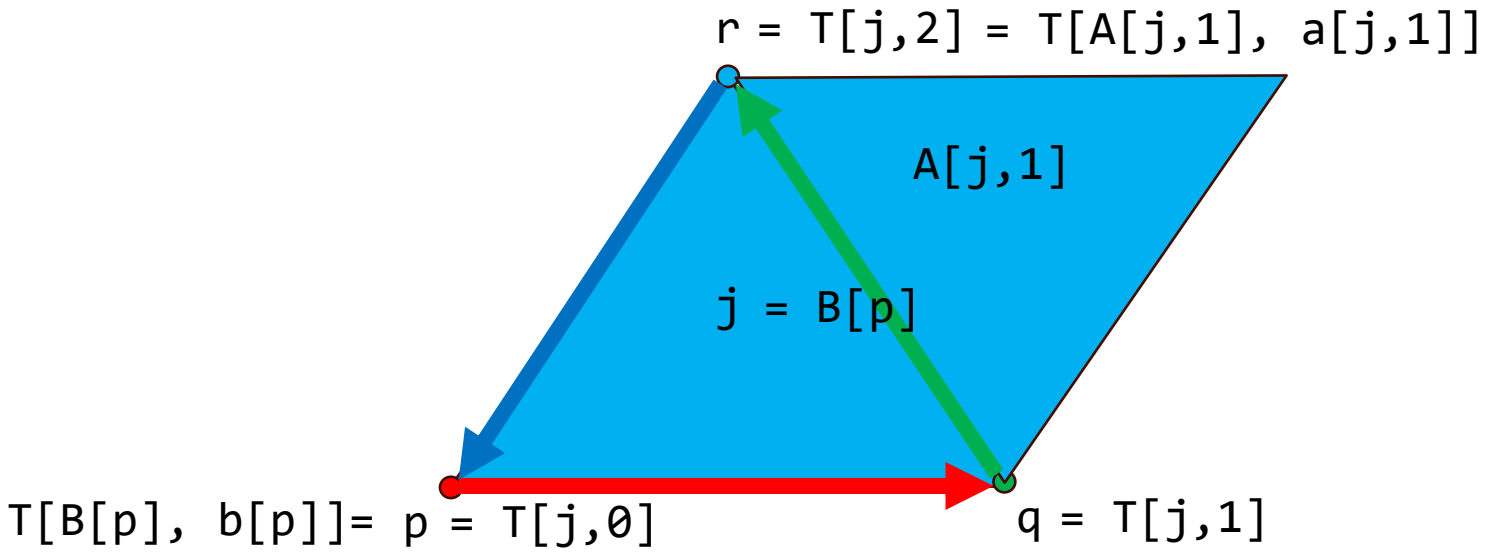
Eliminate edges



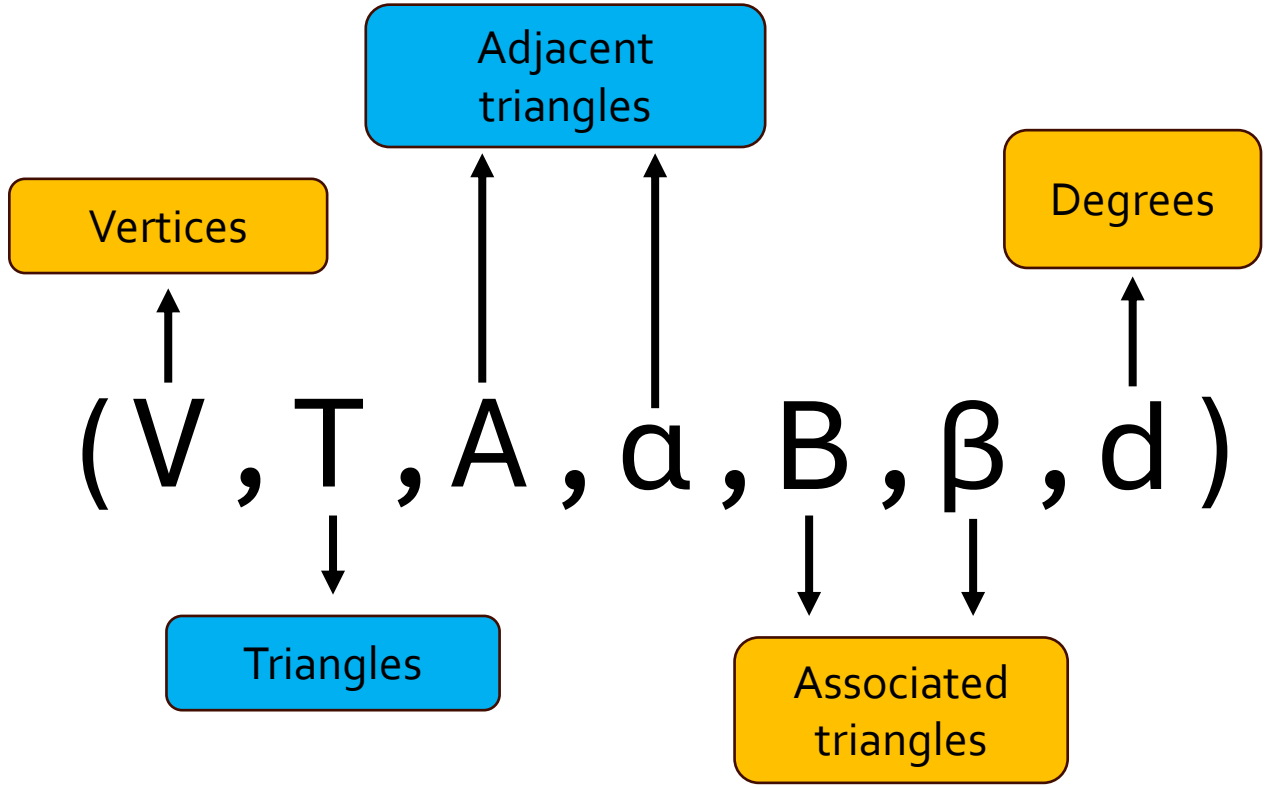
Designing data structure 2/3



Creating „self-references”



Designing a data structure 3/3



Vertices, triangles, edges

>> Set of vertex indices

$$I = \{0, \ldots, n-1\}$$

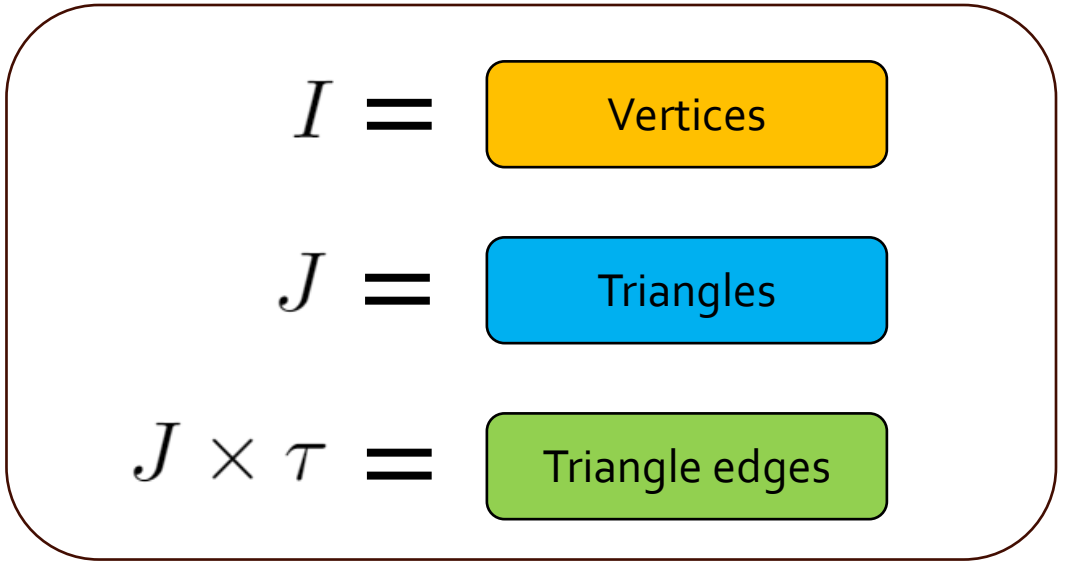
>> Set of triangle indices

$$J = \{0, \ldots, m-1\}$$

>> Set of triangle edge indices

$$\tau = \{0, 1, 2\}$$

$$\forall j \in \tau \quad \forall k \in \mathbb{Z} \ : \ j \oplus k := (j + k) \mod 3$$



SolidMesh definition

$$V : I \to \mathbb{R}^3$$

Vertex coordinates

$$T : J \times \tau \to I^3$$

Triangles/Edges → Vertices

$$A : J \times \tau \to J$$

$$\alpha : J \times \tau \to \tau$$

Triangles/Edges → Edges

Triangles/Edges → Triangles

$$B : I \to J$$

$$\beta : I \to \tau$$

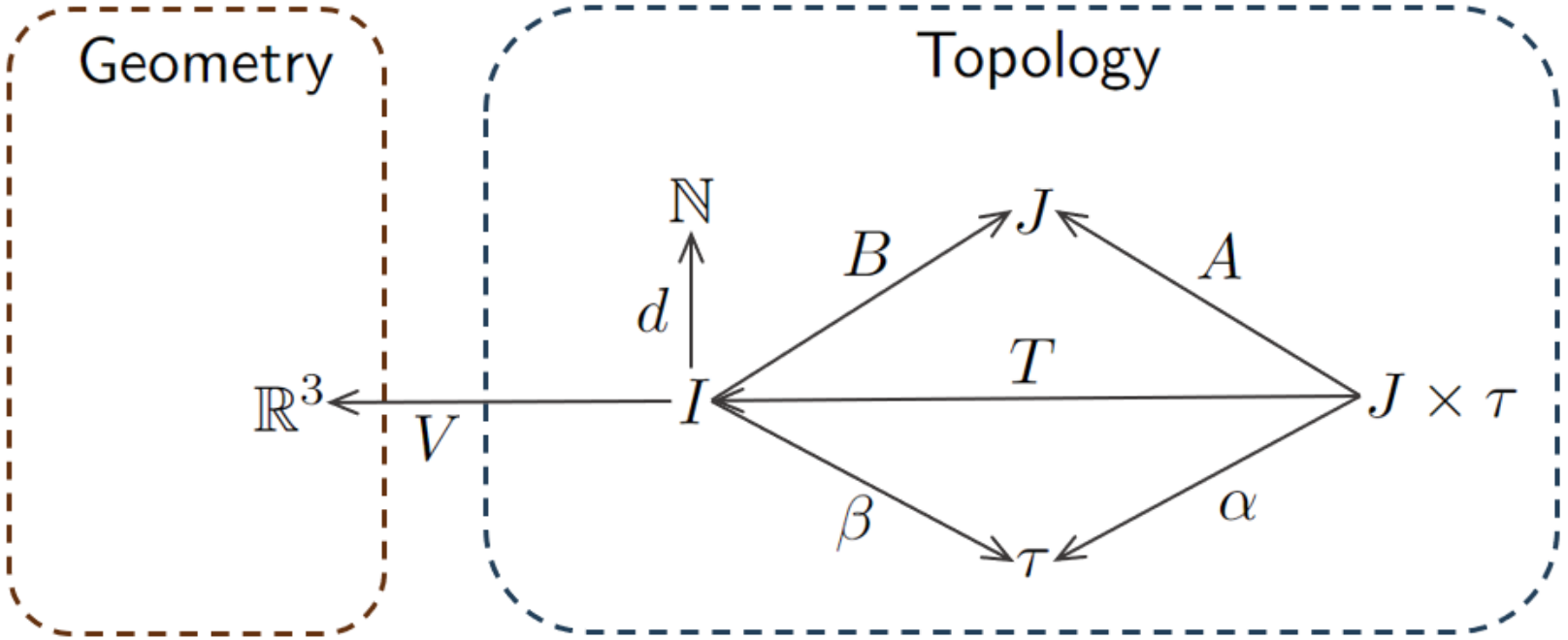
Vertices → Edges

Vertices → Triangles

$$d : I \to \mathbb{N}$$

Vertex degrees

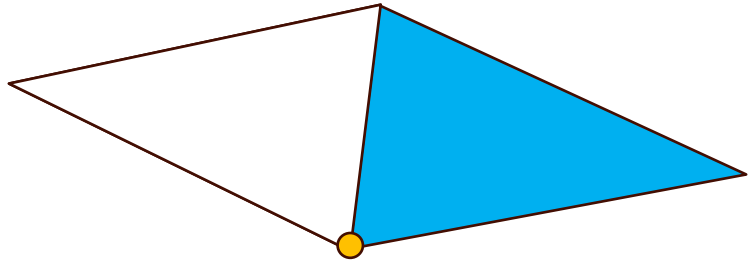
Functions, domains, ranges



Self references in action

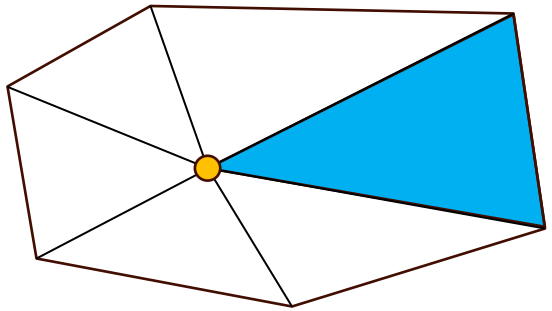
>> The same vertex from the opposite triangle's point of view

$$T(j,k) = T(A(j,k),\ \alpha(j,k) \oplus 1)$$



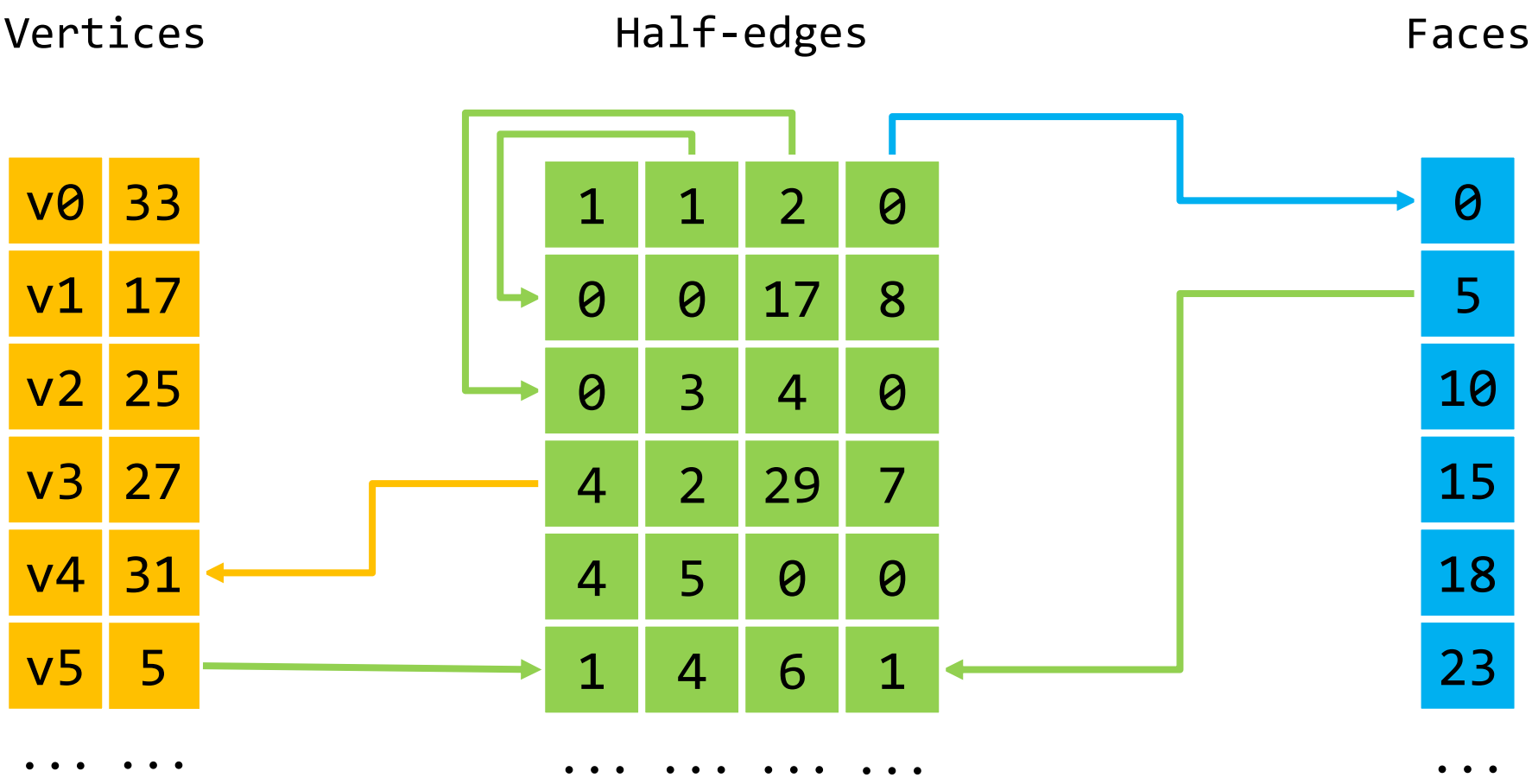
>> The same vertex from the associated triangle's point of view.

$$T(B(i), \beta(i)) = i$$

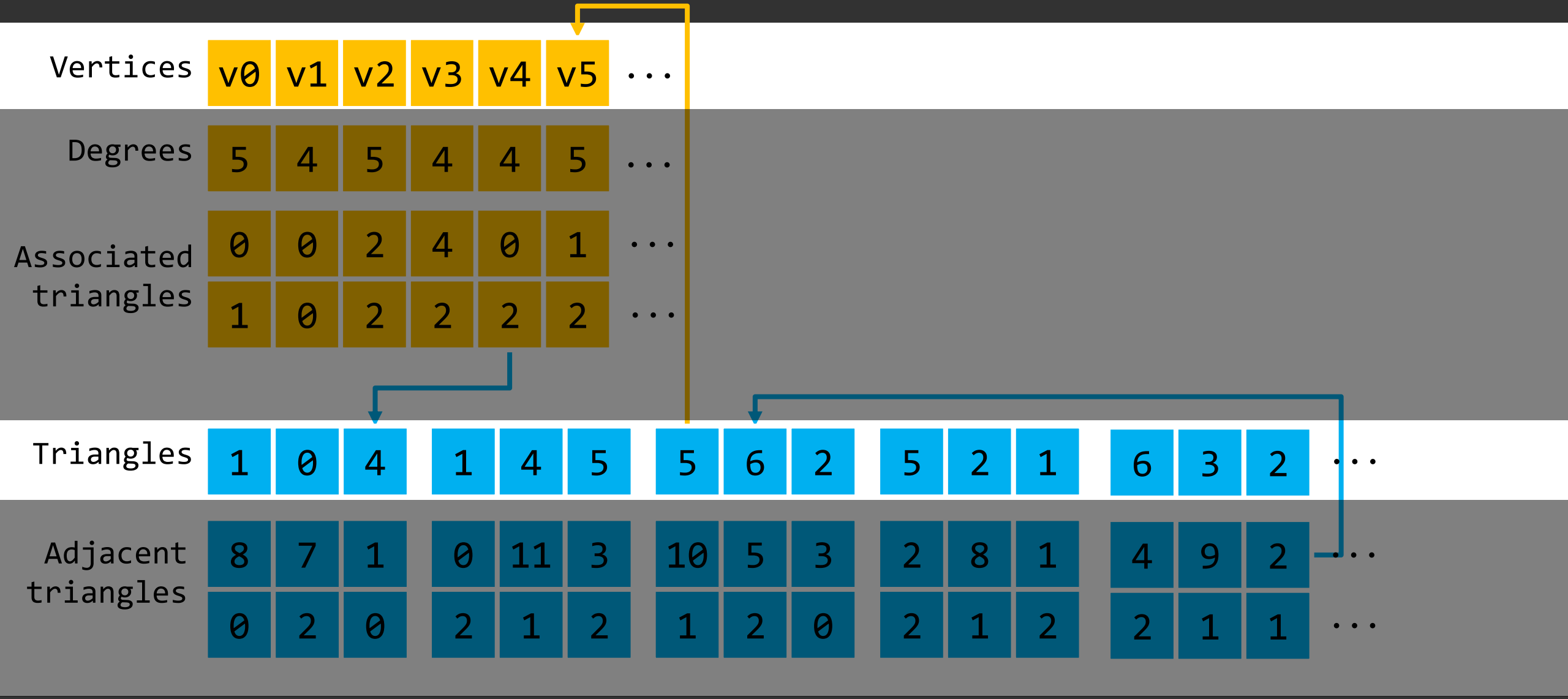


Implementation

Half-edge data structure



SolidMesh data structure



Tests

>> SolidMesh implementation in C# using Unity platform

https://github.com/robagnaibaf/SolidMesh

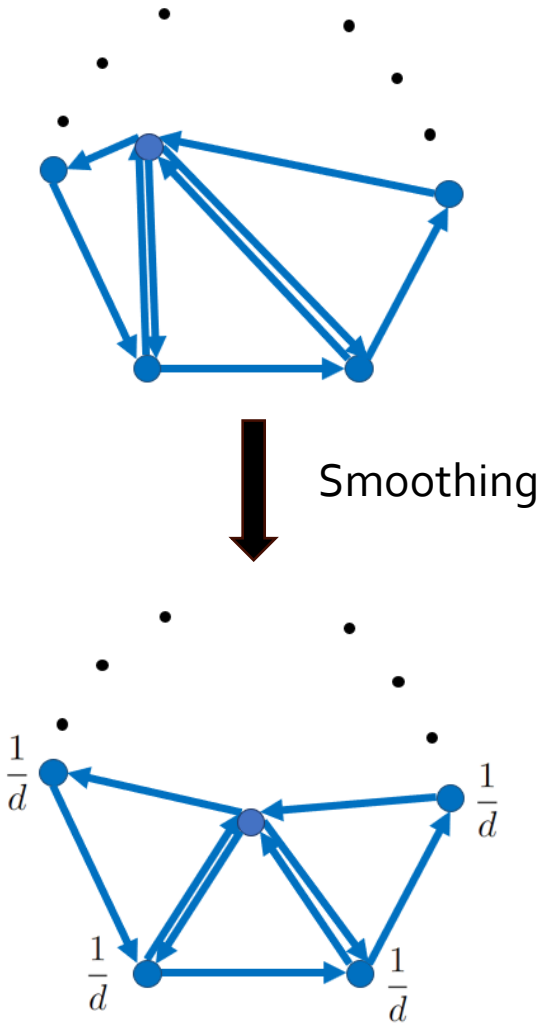
>> Efficient half-edge implementation in C# (ported to Unity)

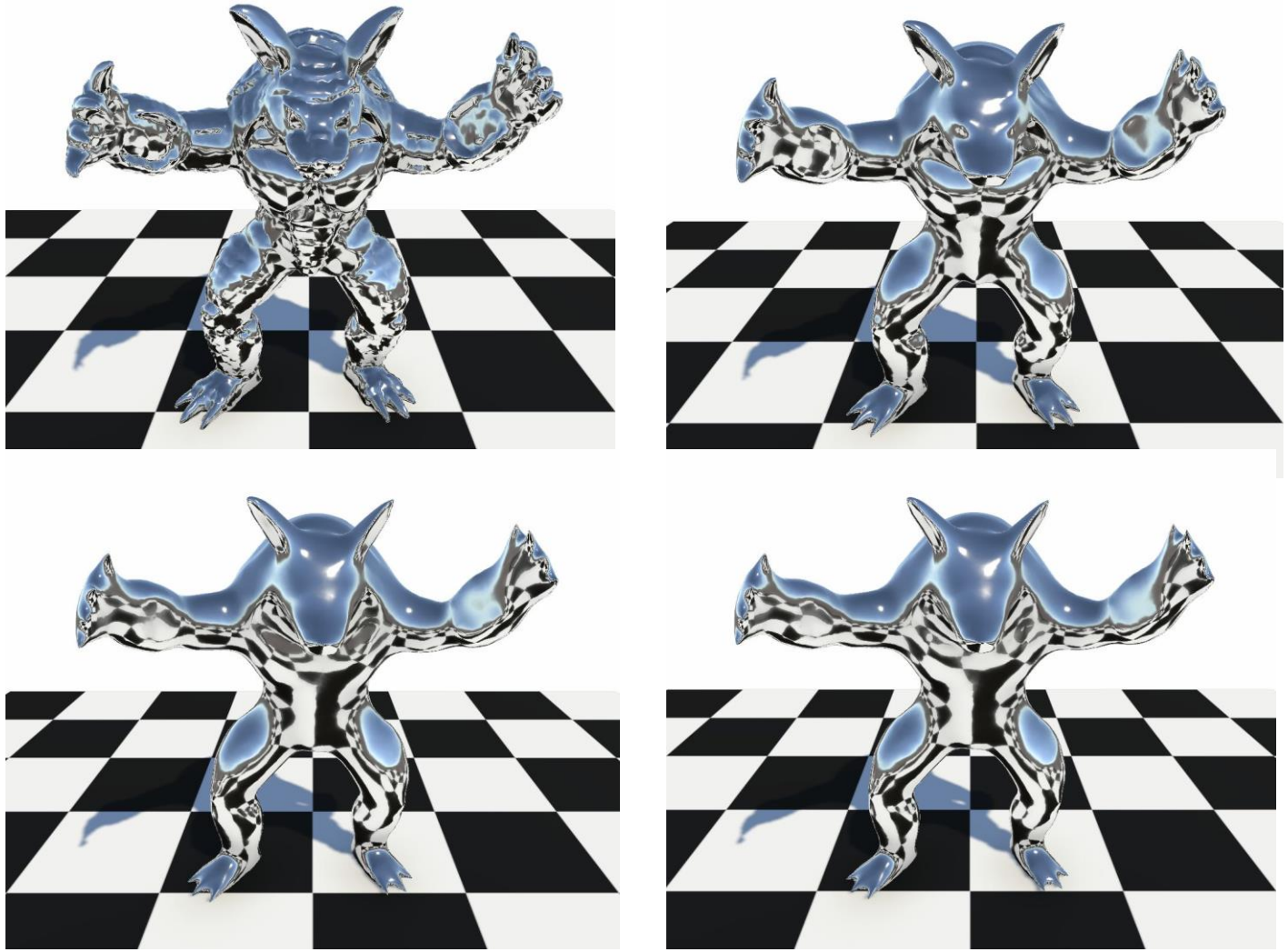
https://github.com/meshmash/Plankton

>> 4 Tests

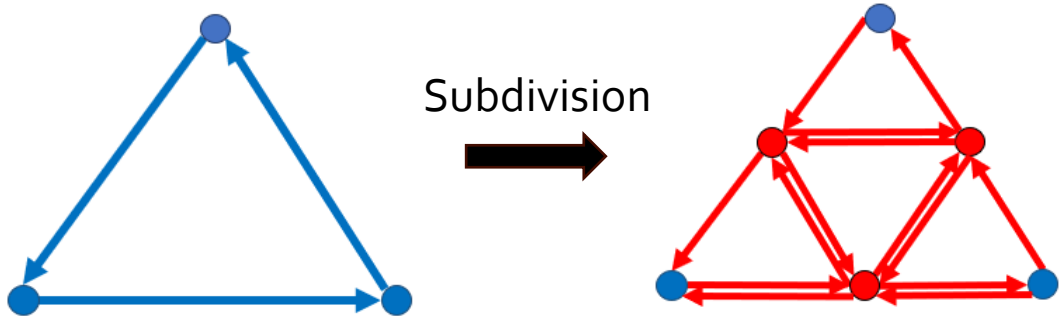
- >> Create data structure from .obj file
- >> Send VertexArray and IndexArray to the GPU
- >> Laplacian smoothing
- >> Loop subdivision

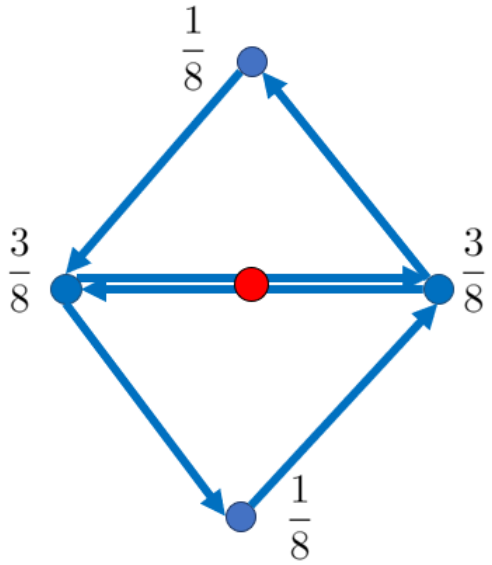
Global operation – Laplacian smoothing

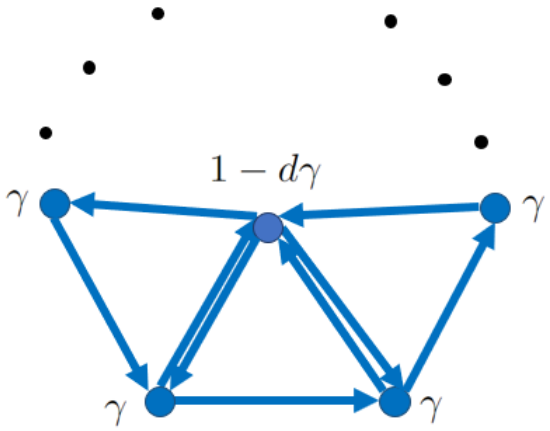


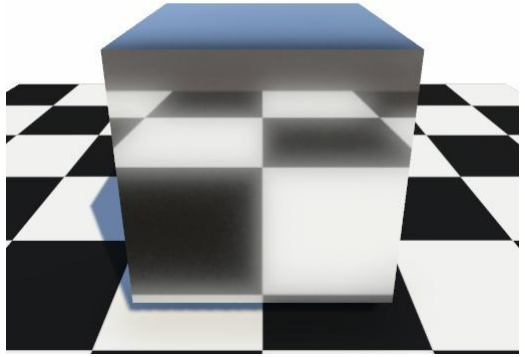


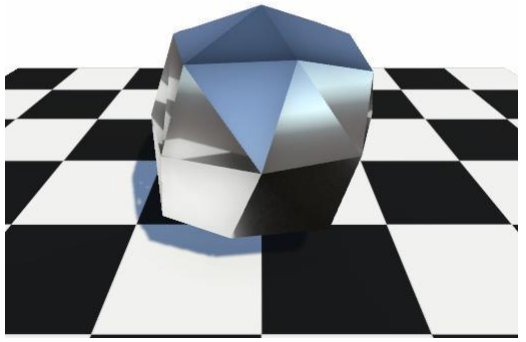
Global operation – Loop subdivision

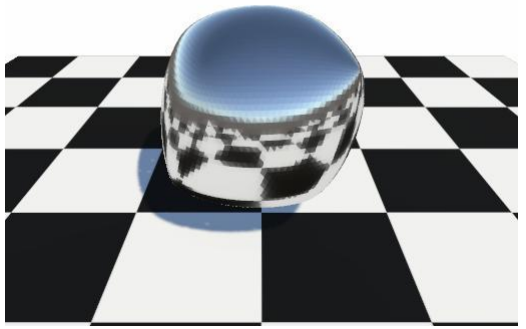










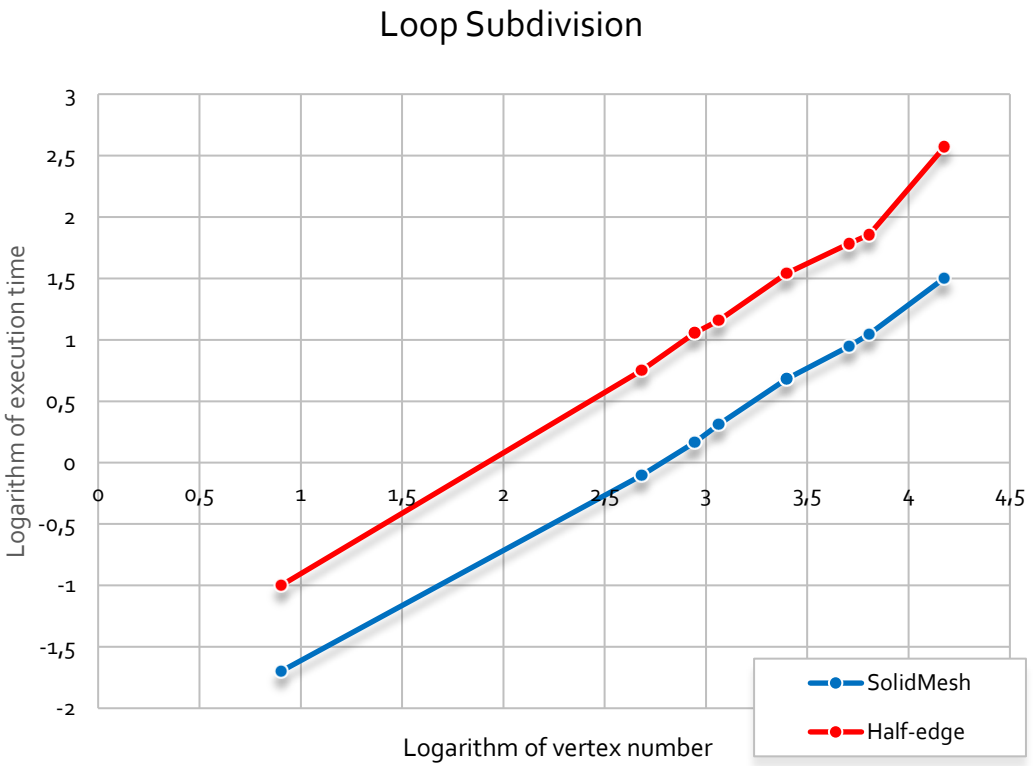


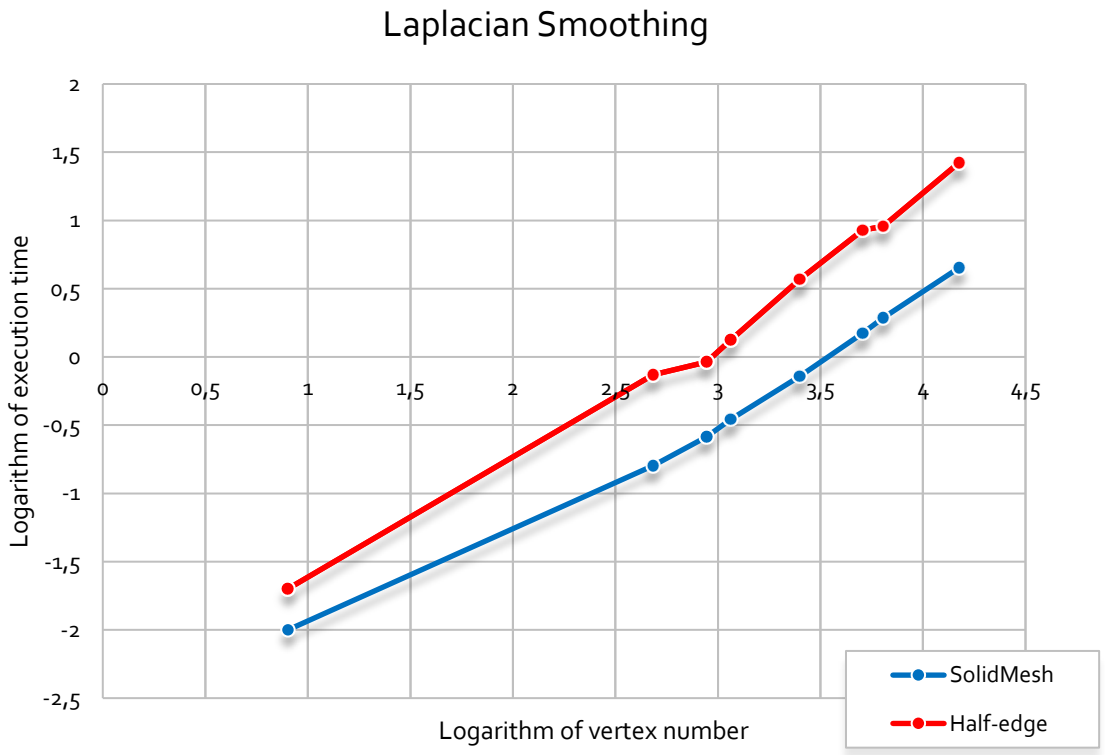
Results – Time costs

Model	Vertices	Creation time [ms]		Rendering time [ms]	
		SolidMesh	Half-edge	SolidMesh	Half-edge
Cube	8	0.04	0.04	**0.04**	0.06
Sphere	482	**1.02**	1.17	**0.09**	0.72
Torusknot	880	**2.02**	2.57	**0.10**	1.21
2-tori	1156	**2.72**	2.85	**0.11**	1.02
Bunny	2503	**6.32**	7.86	**0.27**	3.59
Mug	5084	**12.95**	15.67	**0.30**	4.66
Ducky	6390	**17.48**	18.39	**0.34**	5.43
Armadillo	15002	236.97	**52.97**	**0.84**	14.20

Model	Vertices	Smoothing time [ms]		Subdivision time [ms]	
		SolidMesh	Half-edge	SolidMesh	Half-edge
Cube	8	**0.01**	0.02	**0.02**	0.10
Sphere	482	**0.16**	0.74	**0.79**	5.63
Torusknot	880	**0.26**	0.92	**1.46**	11.36
2-tori	1156	**0.35**	1.33	**2.05**	14.37
Bunny	2503	**0.72**	3.70	**4.81**	34.78
Mug	5084	**1.49**	8.48	**8.83**	60.87
Ducky	6390	**1.93**	9.05	**11.12**	71.99
Armadillo	15002	**4.50**	26.50	**31.74**	370.69

Results – Smoothing & Subdivision





Results – Storage

>> Let us suppose, a mesh has n vertices, m triangles and l edges.

>> For low genus surfaces $m \approx 2n$ and $l \approx 3n$.

>> Vertex coordinates are represented by `float` numbers (12 byte per vertex).

>> Indices (indexing vertices, triangles, half-edges, triangle edges) are represented by `int` numbers (4 byte per index).

Half-edge	SolidMesh
120 bytes / vertex	75 ~~96~~ bytes / vertex

>> Triangle edge indices can be represented by `byte` numbers.

Limitations

>> Better than Half-edge?

>> Depends

- >> Better performance for global operations
- >> For local operations Half-edge is more efficient
- >> Only works for triangle meshes
- >> Only works for closed meshes

Future work

>> Better than Half-edge?

>> Depends

>> Better performance for global operations

>> For local operations Half-edge is more efficient

>> Only works for triangle meshes

>> Only works for closed meshes

Late recover for efficient local operations.

Straightforward formulation for quad meshes

Boundary detection

Attaching / Detaching

Gluing meshes along their boundaries

Thank you for your attention!