SIGGRAPH 2021

RXMesh: A GPU Mesh Data Structure

Ahmed H. Mahmoud[1,2], Serban D. Porumbescu[1], and John D. Owens[1]

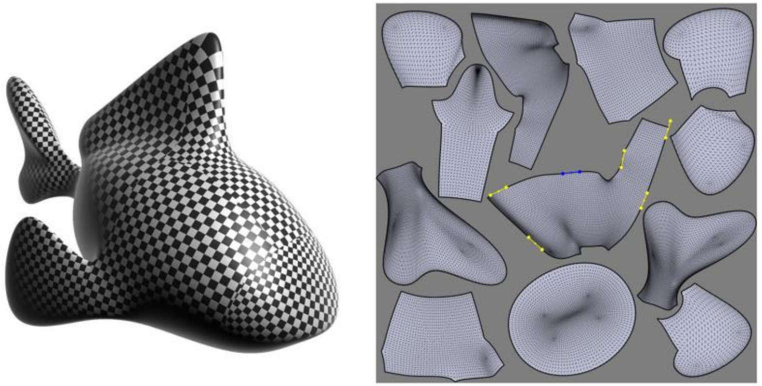
[1]University of California, Davis; [2]Autodesk Research

UC DAVIS

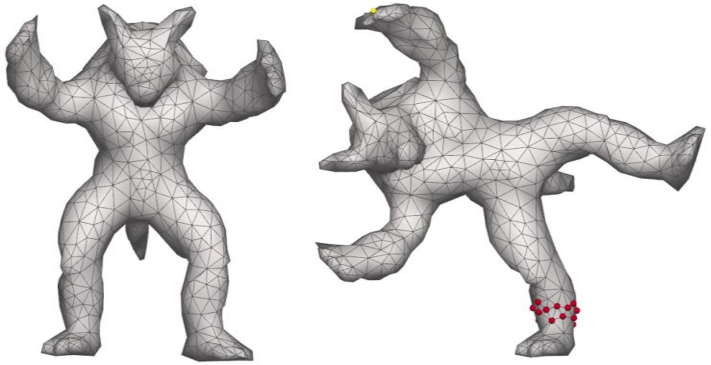
AUTODESK RESEARCH

Motivation

- Triangle meshes are everywhere

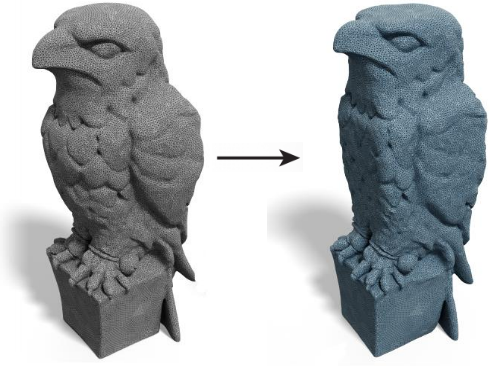


Boundary First Flattening [Sawhney et al., 2018]

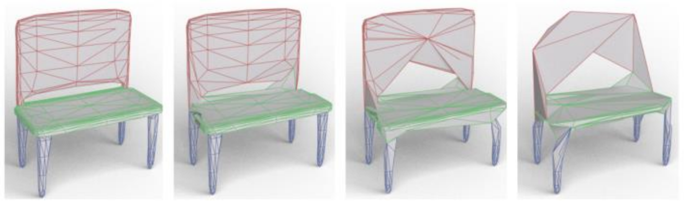


As-Rigid-As-Possible Surface Modeling
[Sorkine et al., 2007]

Adaptive Anisotropic Remeshing for Cloth Simulation [Narain et al., 2012]



Cubic Stylization [Liu et al., 2019]



MeshCNN- A Network with an Edge
[Hanocka et al., 2019]

Motivation

- Triangle meshes are everywhere
- Most of the mesh processing libraries are on the CPUs

Motivation

- Triangle meshes are everywhere
- Most of the mesh processing libraries are on the CPUs
- How to leverage the GPU massive parallelism for mesh processing?

Motivation

SIGGRAPH 2021

- Triangle meshes are everywhere
- Most of the mesh processing libraries are on the CPUs
- How to leverage the GPU massive parallelism for mesh processing?

Programming Model

- Intuitive and simple
- High-level abstraction

Data Structure

- High performance
- Generic
- Compact

RXMesh Programming Model

RXMesh Programming Model
Motivation

- Examples of GPU-specific programming models
  - Image processing [Halide: Ragan-Kelley et al. 2013]
  - Sparse voxel computation [Taichi: Hu et al. 2019]
  - Simulation [Ebb: Bernstein et al. 2016]
  - Graph processing [Gunrock: Wang et al. 2017]

RXMesh Programming Model

- Focuses only on applications that require local computation

RXMesh Programming Model

SIGGRAPH 2021

- Focuses only on applications that require local computation

- Requires the user to think only of the operations applied locally to a single mesh element
  - Neighbor queries
  - Attributes queries

RXMesh Programming Model

- Focuses only on applications that require local computation
- Requires the user to think only of the operations applied locally to a single mesh element
  - Neighbor queries
  - Attributes queries
- Inspired by Think-Like-A-Vertex [McCune et al. 2015] programming model for graph processing

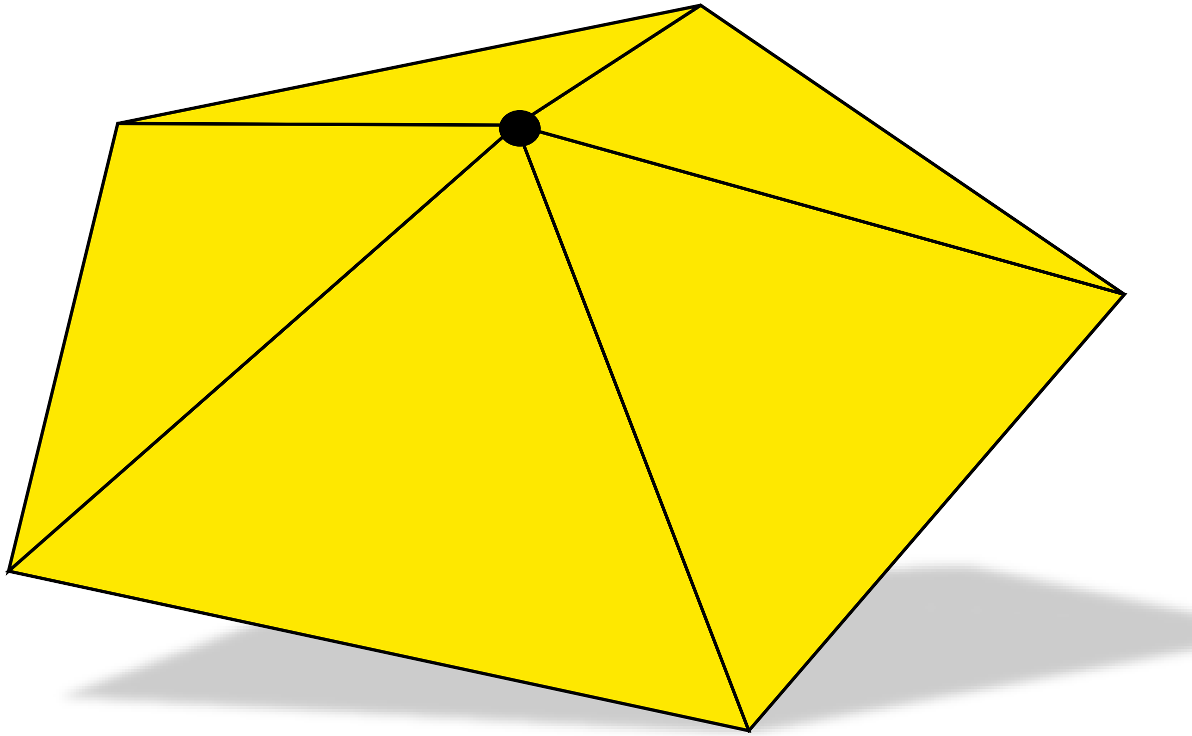
RXMesh Programming Model

SIGGRAPH 2021

- Focuses only on applications that require local computation
- Requires the user to think only of the operations applied locally to a single mesh element
  - Neighbor queries
  - Attributes queries
- Inspired by Think-Like-A-Vertex [McCune et al. 2015] programming model for graph processing
- We extend it to all mesh elements i.e., vertices, edges, and faces

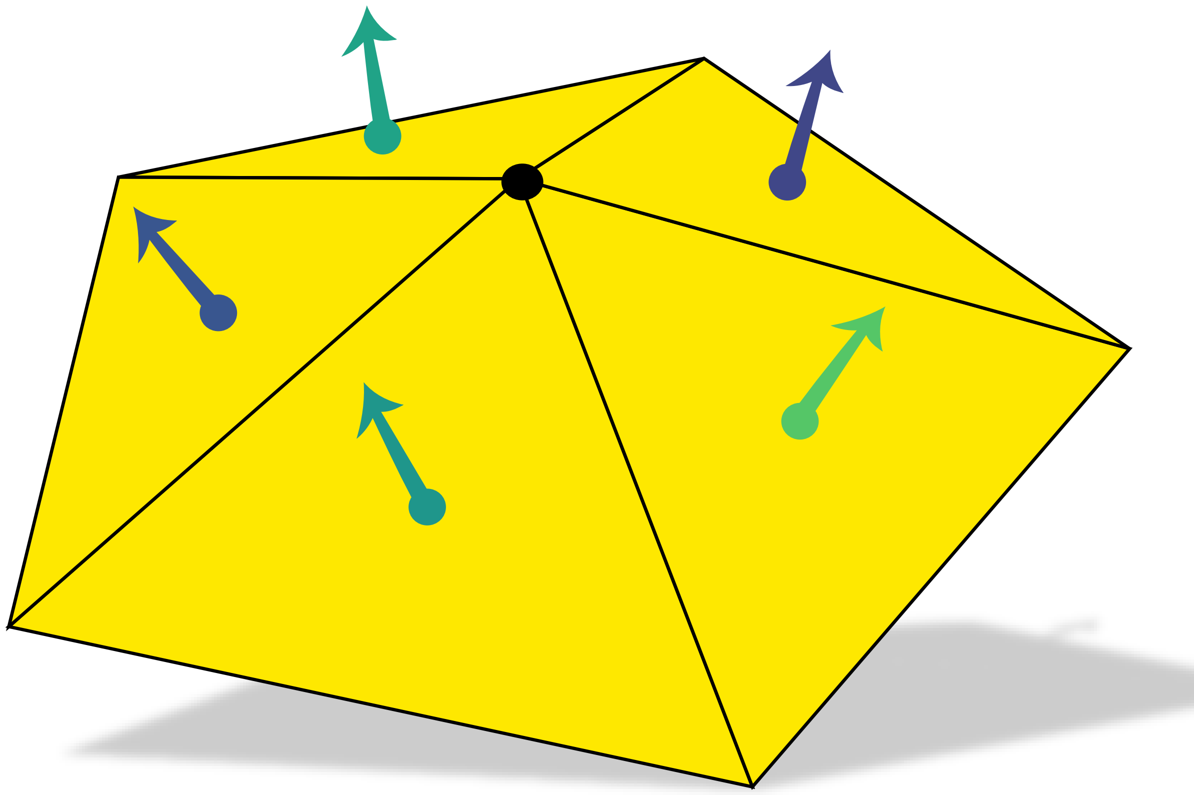
RXMesh Programming Model

- Vertex normal computation



RXMesh Programming Model

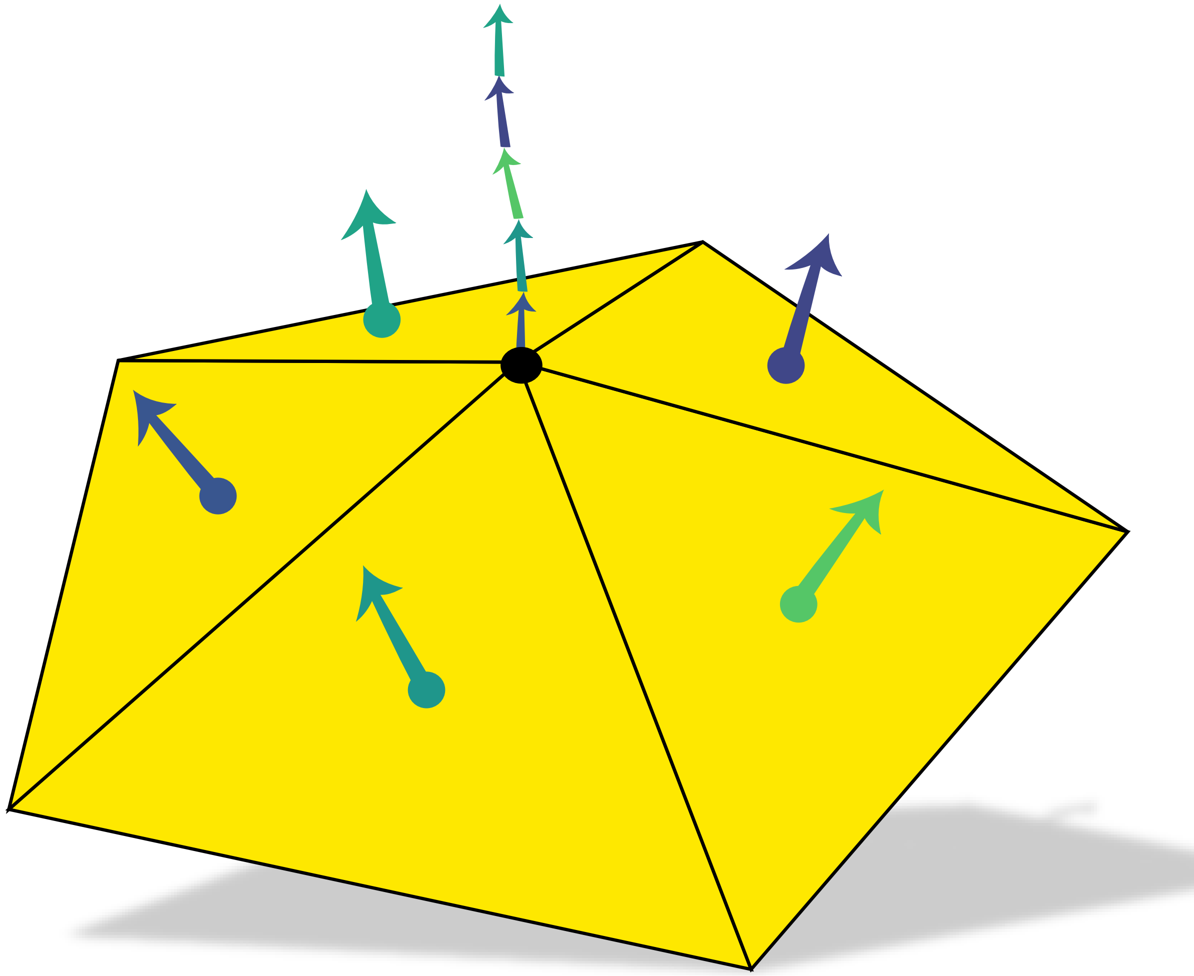
- Vertex normal computation



RXMesh Programming Model

SIGGRAPH 2021

- Vertex normal computation
  1. Query the face's three vertices
  2. Compute the face's normal
  3. Atomically add the face's normal to its vertices



RXMesh Programming Model

- Vertex normal computation

```
__global__ void
  ComputeVertexNormal(RXMesh                  rxmesh,
                      Vec3<float>*          VertexNormals,
                      const Vec3<float>*    VertexCoords) {
  rxmesh.template kernel<Op::FV>(
    [&](const uint32_t f_id, const Iterator fv_iter){
    //The face's three vertices
    uint32_t v0(fv_iter[0]), v1(fv_iter[1]), v2(fv_iter[2]);

    //Compute face normal
    Vec3<float> faceNormal = ComputeFaceNormal(v0, v1, v2,
                                               VertexCoords);
    //Update vertex normals with faceNormal component
    atomicAdd<Vec3<float>>(VertexNormals[v0], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v1], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v2], faceNormal);});
}
```

RXMesh Programming Model

- Vertex normal computation

```
__global__ void
  ComputeVertexNormal(RXMesh                  rxmesh,
                      Vec3<float>*            VertexNormals,
                      const Vec3<float>*      VertexCoords) {
  rxmesh.template kernel<Op::FV>(
    [&](const uint32_t f_id, const Iterator fv_iter){
    //The face's three vertices
    uint32_t v0(fv_iter[0]), v1(fv_iter[1]), v2(fv_iter[2]);

    //Compute face normal
    Vec3<float> faceNormal = ComputeFaceNormal(v0, v1, v2,
                                               VertexCoords);
    //Update vertex normals with faceNormal component
    atomicAdd<Vec3<float>>(VertexNormals[v0], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v1], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v2], faceNormal);});
}
```

RXMesh Programming Model

SIGGRAPH 2021

- Vertex normal computation

```
__global__ void
  ComputeVertexNormal(RXMesh              rxmesh,
                      Vec3<float>*        VertexNormals,
                      const Vec3<float>*  VertexCoords) {
  rxmesh.template kernel<Op::FV>(
    [&](const uint32_t f_id, const Iterator fv_iter){
    //The face's three vertices
    uint32_t v0(fv_iter[0]), v1(fv_iter[1]), v2(fv_iter[2]);

    //Compute face normal
    Vec3<float> faceNormal = ComputeFaceNormal(v0, v1, v2,
                                               VertexCoords);
    //Update vertex normals with faceNormal component
    atomicAdd<Vec3<float>>(VertexNormals[v0], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v1], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v2], faceNormal);});
}
```

RXMesh Programming Model

- Vertex normal computation

```
__global__ void
  ComputeVertexNormal(RXMesh             rxmesh,
                      Vec3<float>*       VertexNormals,
                      const Vec3<float>* VertexCoords) {
  rxmesh.template kernel<Op::FV>(
    [&](const uint32_t f_id, const Iterator fv_iter){
    //The face's three vertices
    uint32_t v0(fv_iter[0]), v1(fv_iter[1]), v2(fv_iter[2]);

    //Compute face normal
    Vec3<float> faceNormal = ComputeFaceNormal(v0, v1, v2,
                                               VertexCoords);
    //Update vertex normals with faceNormal component
    atomicAdd<Vec3<float>>(VertexNormals[v0], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v1], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v2], faceNormal);});
}
```

RXMesh Programming Model

- Vertex normal computation

```
__global__ void
  ComputeVertexNormal(RXMesh                rxmesh,
                      Vec3<float>*          VertexNormals,
                      const Vec3<float>*    VertexCoords) {
  rxmesh.template kernel<Op::FV>(
    [&](const uint32_t f_id, const Iterator fv_iter){
    //The face's three vertices
    uint32_t v0(fv_iter[0]), v1(fv_iter[1]), v2(fv_iter[2]);

    //Compute face normal
    Vec3<float> faceNormal = ComputeFaceNormal(v0, v1, v2,
                                               VertexCoords);
    //Update vertex normals with faceNormal component
    atomicAdd<Vec3<float>>(VertexNormals[v0], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v1], faceNormal);
    atomicAdd<Vec3<float>>(VertexNormals[v2], faceNormal);});
}
```

RXMesh Programming Model

SIGGRAPH 2021

User's Responsibility

- Define computation that run on a single mesh element

Programming Model's Responsibility

- Run computation on all mesh elements
- Assign GPU threads to mesh elements
- Maximize locality
- Induce load balance

SIGGRAPH 2021

RXMesh Data Structure:

- Design Goals
- Design Principles

What queries RXMesh supports

Query	Definition
VV	For vertex V, return adjacent vertices
VE	For vertex V, return incident edges
VF	For vertex V, return incident faces
EV	For edge E, return incident vertices
EF	For edge E, return incident faces
FV	For face F, return incident vertices
FE	For face F, return incident edges
FF	For face F, return adjacent faces

Design Goals

1. Performance

- Improve locality and confine computation within the shared memory

Design Goals

SIGGRAPH 2021

1. **Performance**
   - Improve locality and confine computation within the shared memory
2. **Generality**
   - No assumption on mesh quality e.g., non-manifold
   - Operates on vertices, edges, and faces

Design Goals

SIGGRAPH 2021

1. Performance

- Improve locality and confine computation within the shared memory

2. Generality

- No assumption on mesh quality e.g., non-manifold
- Operate on vertices, edges, and faces

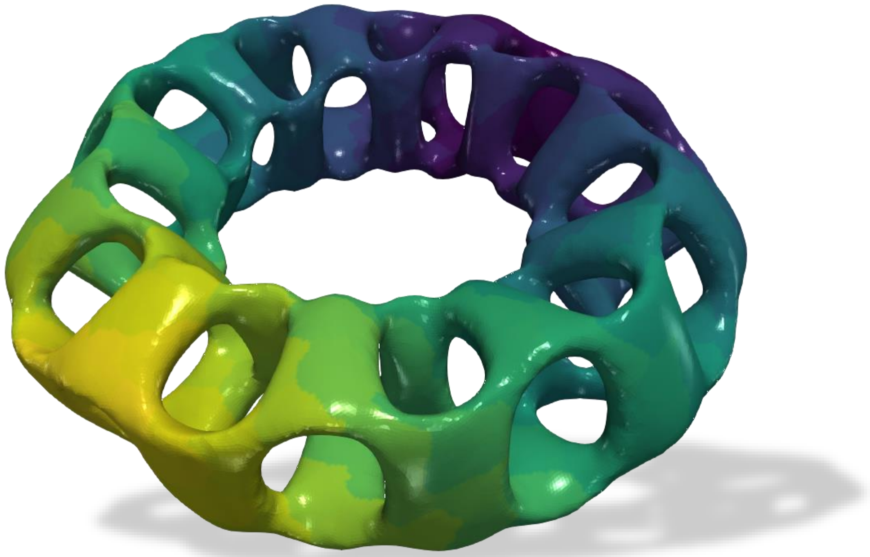
3. Compactness

- Store minimal amount of data and compute query on-the-fly

Design Principles

SIGGRAPH 2021

1. Locality by Patching



Global Sorting

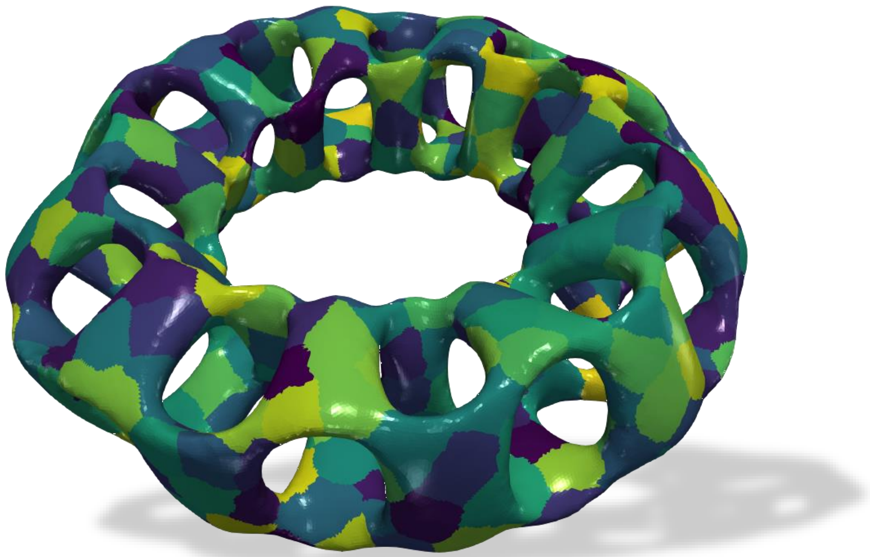
Color indicates face index

Design Principles

1. Locality by Patching

Global Sorting

Color indicates face index



Patching

Color indicates patch ID

Design Principles

SIGGRAPH 2021

2. Patch Representation

- Small patches promote reduced precision i.e., 16-bit

Design Principles

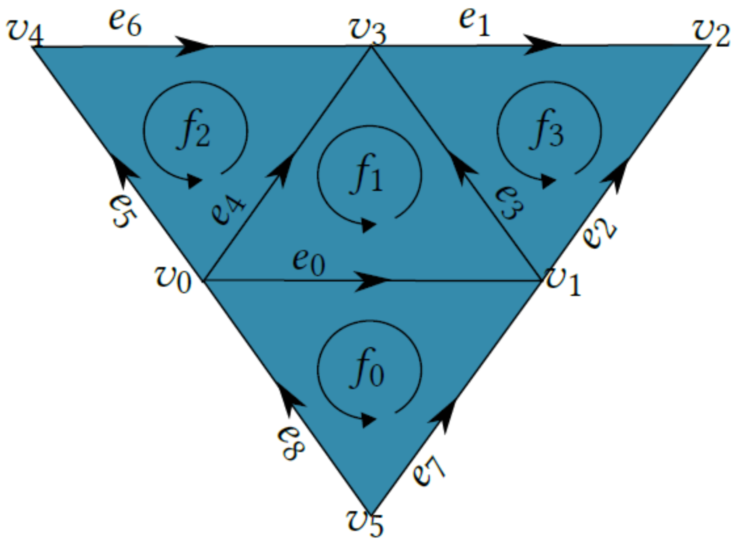
2. Patch Representation

- Small patches promote reduced precision i.e., 16-bit
- Represent patches using Linear Algebraic Representation (LAR) [DiCarlo et al., 2014]

Design Principles

2. Patch Representation

- Small patches promote reduced precision i.e., 16-bit
- Represent patches using Linear Algebraic Representation (LAR) [DiCarlo et al., 2014]

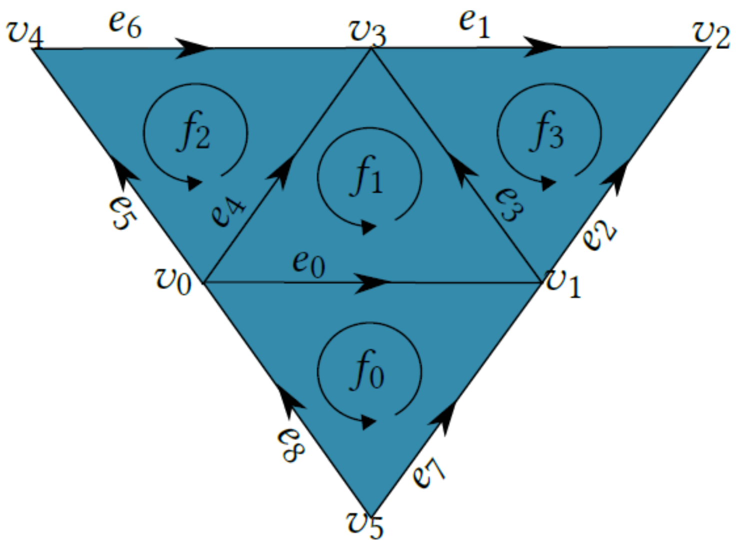


	v_0	v_1	v_2	v_3	v_4	v_5
e_0	-1	1				
e_1			1	-1		
e_2		-1	1			
e_3		-1		1		
e_4	-1			1		
e_5	-1				1	
e_6				1	-1	
e_7		1				-1
e_8	1					-1

M_{EV}

2. Patch Representation

- Small patches promote reduced precision i.e., 16-bit
- Represent patches using Linear Algebraic Representation (LAR) [DiCarlo et al., 2014]



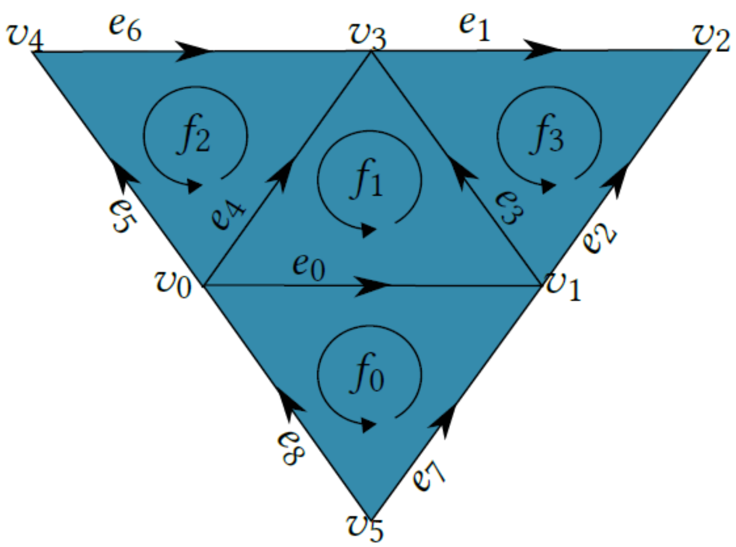
	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
f_0	-2							1	-3
f_1	1			2	-3				
f_2					1	-3	-2		
f_3		-2	1	-3					

M_{FE}

Design Principles

2. Patch Representation

- Small patches promote reduced precision i.e., 16-bit
- Represent patches using Linear Algebraic Representation (LAR) [DiCarlo et al., 2014]



	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_0	1				1	1			1
v_1	1		1	1				1	
v_2		1	1						
v_3		1		1	1		1		
v_4						1	1		
v_5								1	1

$M_{VE} = M_{EV}^T$

	f_0	f_1	f_2	f_3
v_0	1	1	1	
v_1	1	1		1
v_2				1
v_3		1	1	1
v_4			1	
v_5	1			

$M_{VF} = M_{EV}^T M_{FE}^T$

	v_0	v_1	v_2	v_3	v_4	v_5
v_0	1	1		1	1	1
v_1	1	1	1	1		1
v_2		1	1	1		
v_3	1	1	1	1	1	
v_4	1			1	1	
v_5	1	1				1

$M_{VV} = M_{EV}^T M_{EV}$

	f_0	f_1	f_2	f_3
e_0	1	1		
e_1				1
e_2				1
e_3		1		1
e_4		1	1	
e_5			1	
e_6			1	
e_7	1			
e_8	1			

$M_{EF} = M_{FE}^T$

	v_0	v_1	v_2	v_3	v_4	v_5
f_0	1	1				1
f_1	1	1		1		
f_2	1			1	1	
f_3		1	1	1		

$M_{FV} = M_{FE} M_{EV}$

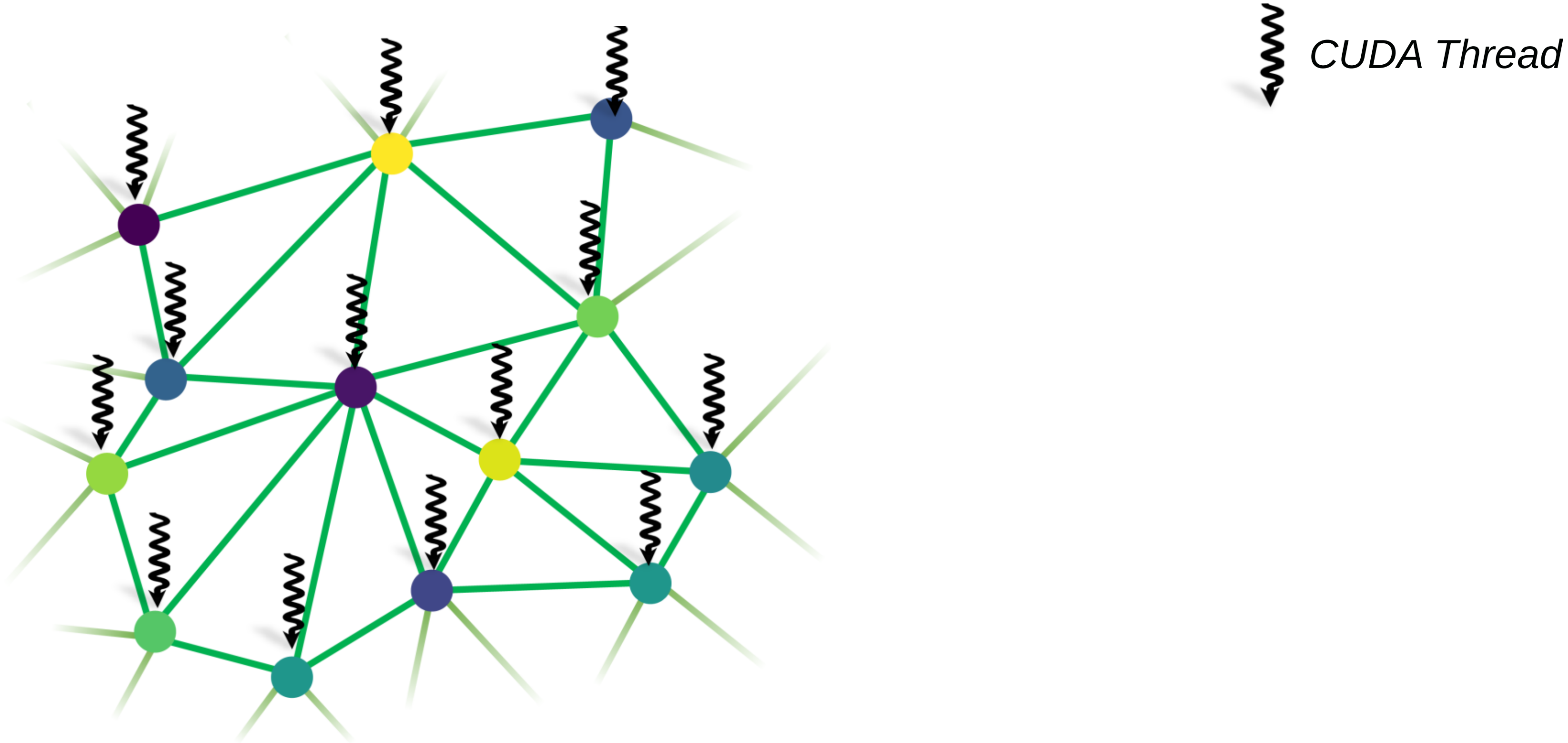
	f_0	f_1	f_2	f_3
f_0	1	1		
f_1	1	1	1	1
f_2		1	1	
f_3		1		1

$M_{FF} = M_{FE} M_{FE}^T$

Design Principles

SIGGRAPH 2021

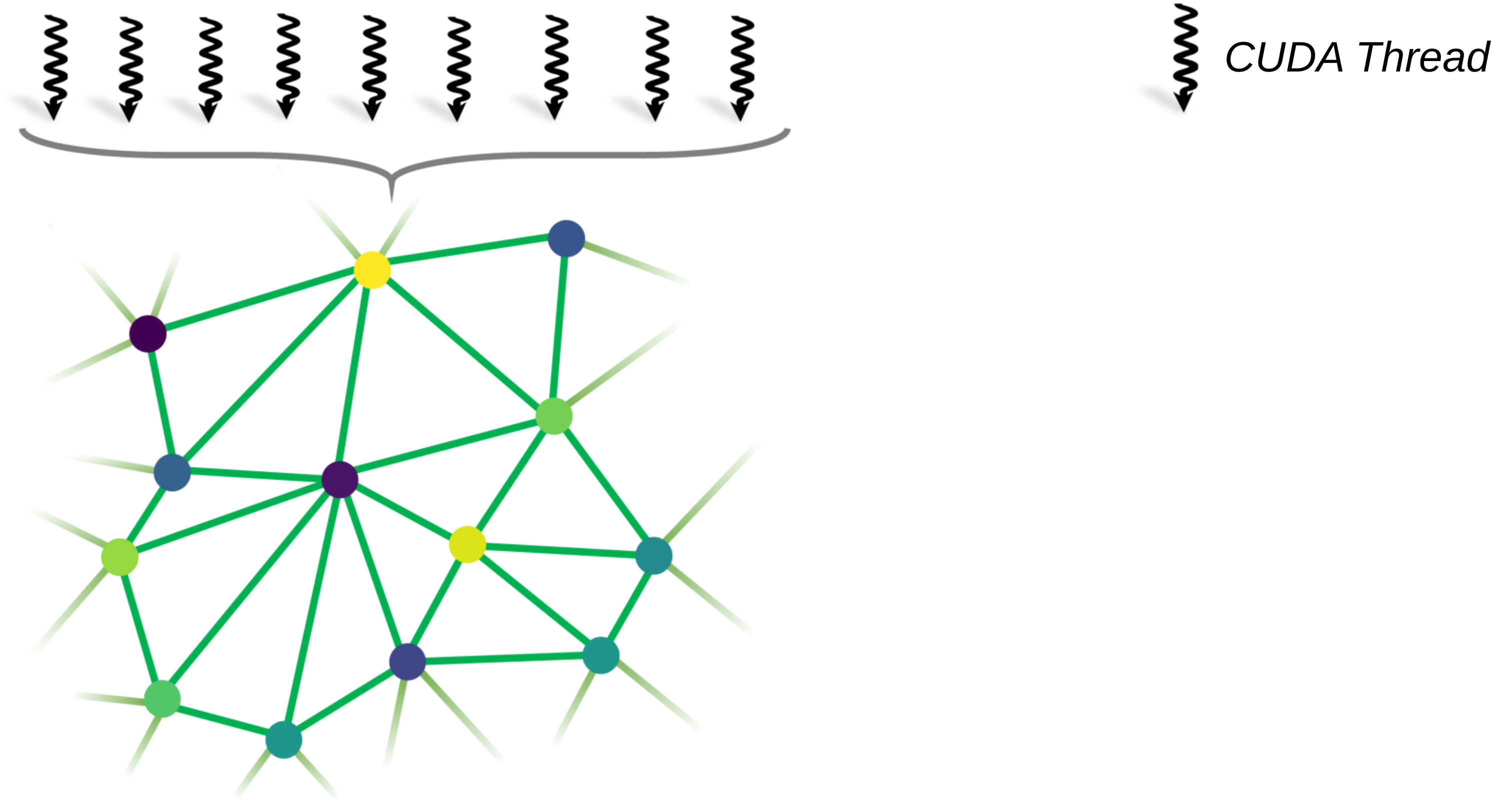
3. Work Mapping



Threads work ***independently***

Design Principles

3. Work Mapping



Threads work ***collaboratively***

Design Principles

SIGGRAPH 2021

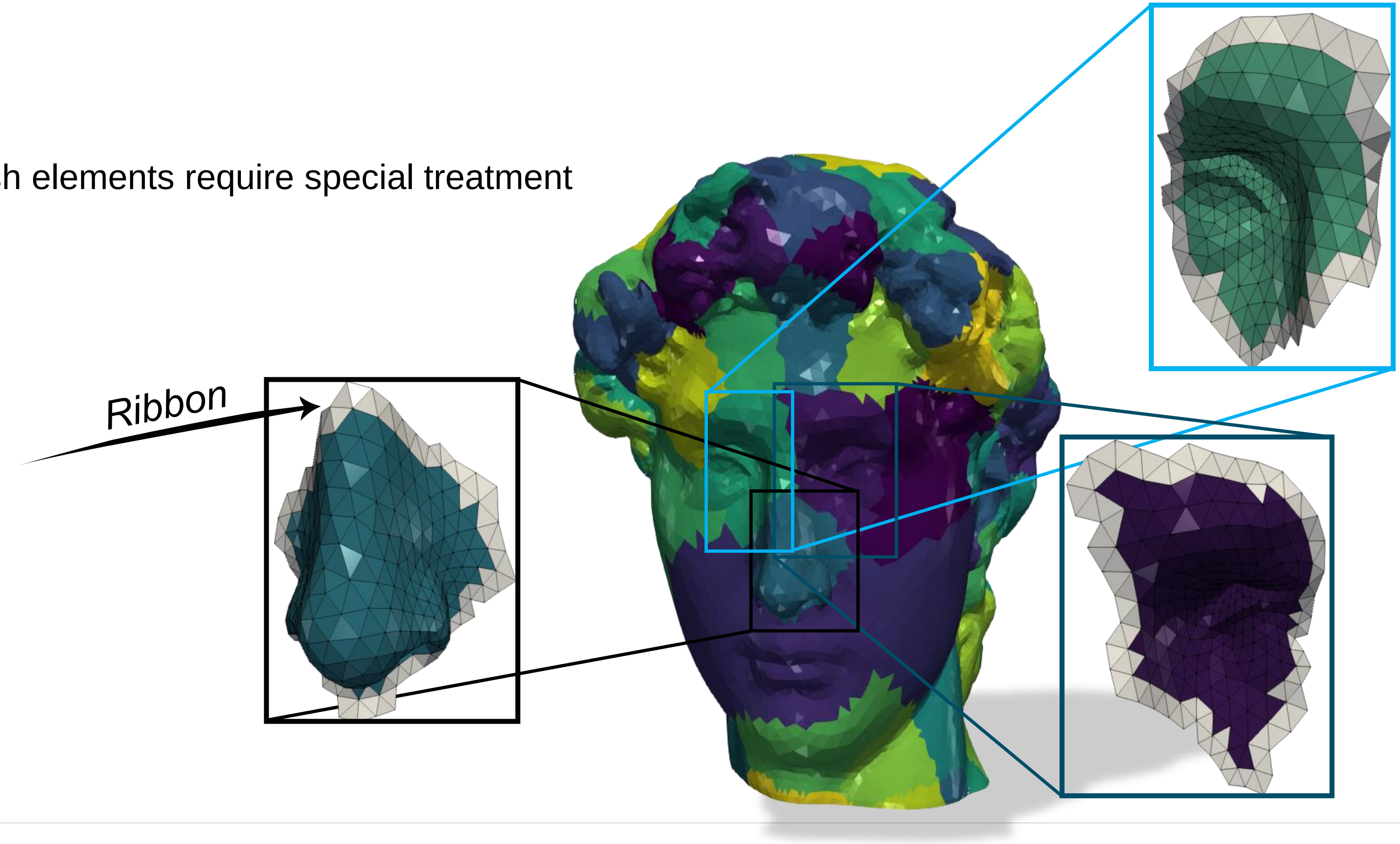
4. Ribbons

- Patch-boundary mesh elements require special treatment

Design Principles

4. Ribbons

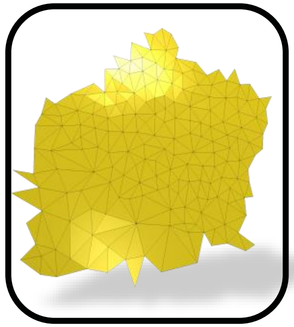
- Patch-boundary mesh elements require special treatment



Design Principles

5. Index Spaces

- *Local index space* to perform query operations
- *Global index space* for convenience



$$M_{FE} = \begin{array}{c|ccccccc} & e_0 & e_1 & e_2 & \cdots & \cdots & e_{n_e-2} & e_{n_e-1} \\ \hline f_0 & 0 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ f_1 & 0 & 0 & 1 & \cdots & \cdots & 0 & 1 \\ \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ \cdots & 0 & 0 & 0 & \cdots & \cdots & 1 & 0 \\ f_{n_f-1} & 0 & 0 & 1 & \cdots & \cdots & 0 & 0 \end{array}$$

Local Index

To global index

$$M_{EV} = \begin{array}{c|ccccccc} & v_0 & v_1 & v_2 & \cdots & \cdots & v_{n_v-2} & v_{n_v-1} \\ \hline e_0 & 0 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ e_1 & 0 & 0 & 1 & \cdots & \cdots & 0 & 1 \\ e_2 & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ \cdots & 0 & 0 & 0 & \cdots & \cdots & 1 & 0 \\ \cdots & 0 & 0 & 1 & \cdots & \cdots & 0 & 0 \\ \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ \cdots & 0 & 0 & 0 & \cdots & \cdots & 1 & 0 \\ e_{n_e-2} & 0 & 0 & 1 & \cdots & \cdots & 0 & 0 \\ e_{n_e-1} & 0 & 0 & 0 & \cdots & \cdots & 0 & 1 \end{array}$$

SIGGRAPH 2021

Evaluation:

1. Query Operations
2. Applications
   - Mean Curvature Flow
   - Geodesic Distance
   - Bilateral Filtering
   - Vertex Normal

Evaluation

SIGGRAPH 2021

- **CPU**
  - Single- and multi-threaded OpenMesh and CGAL
- **GPU**
  - Parallel Directed Edges (PDE) [Campagna et al. 1998]

Evaluation

- **CPU**
  - Single- and multi-threaded OpenMesh and CGAL: *slower by order of magnitude*
- **GPU**
  - Parallel Directed Edges (PDE) [Campagna et al. 1998]

Evaluation

- **Input order**

Evaluation

Color indicates face index

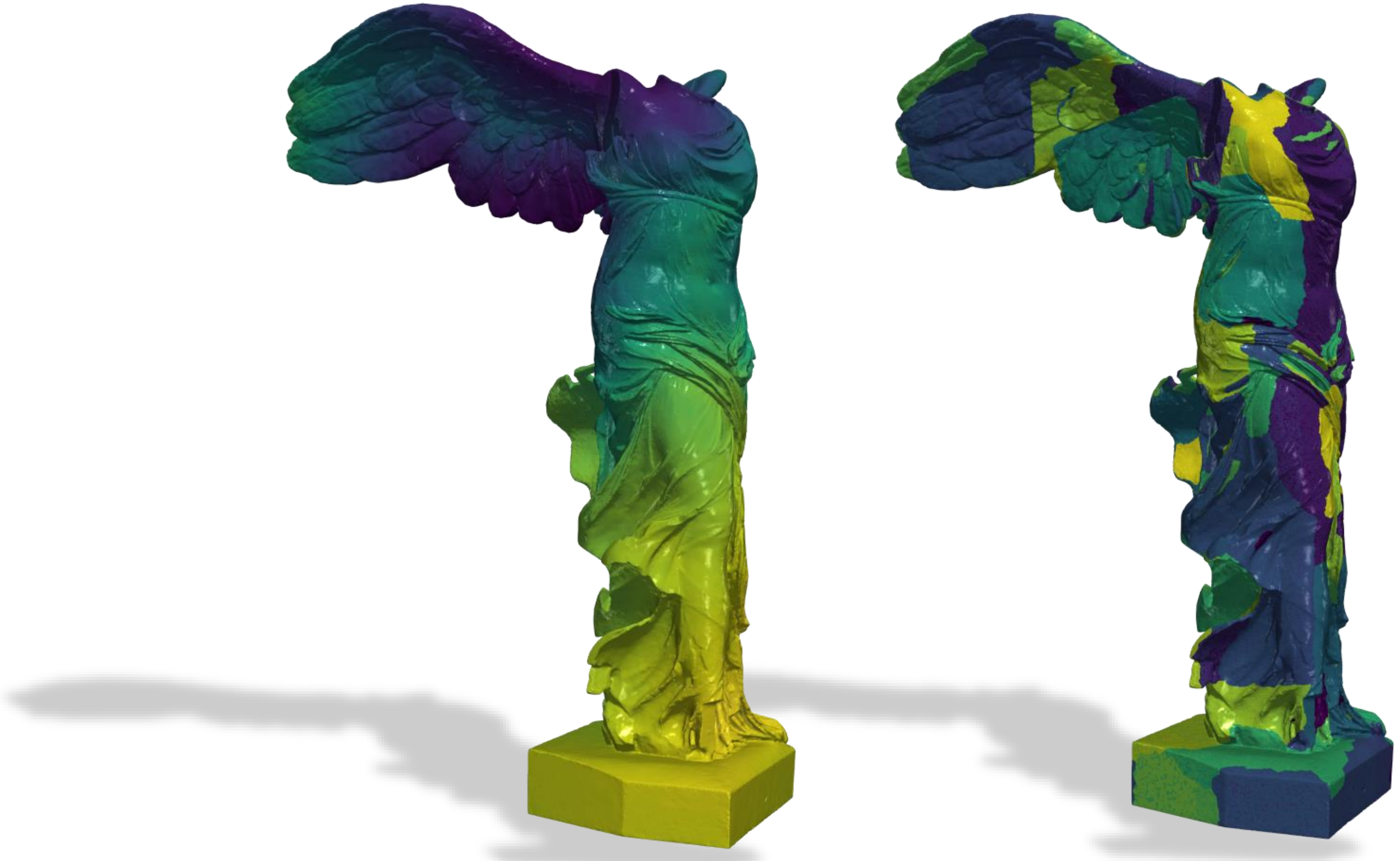
- **Input order**

Default

Evaluation

- **Input order**

Color indicates face index

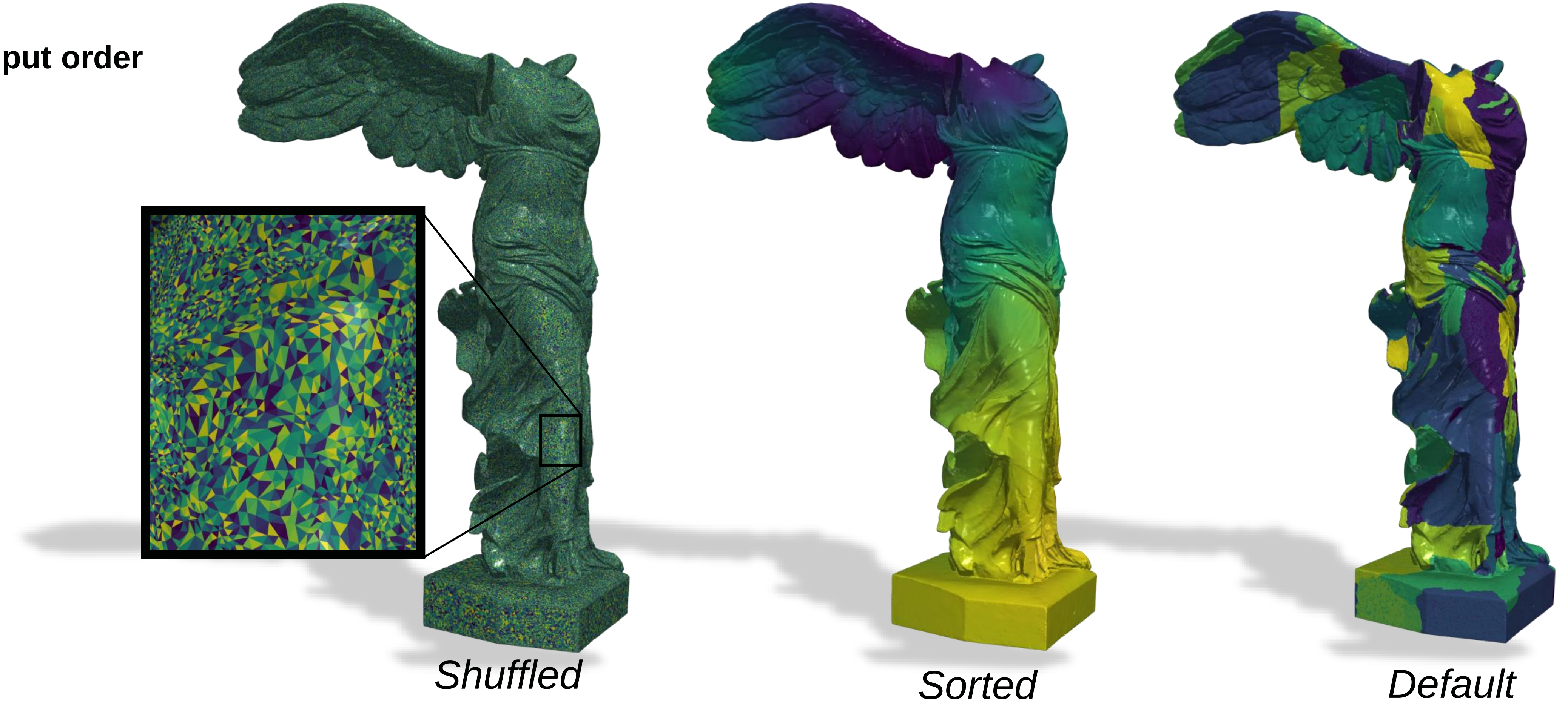


Sorted *Default*

Evaluation

Color indicates face index

- **Input order**



Evaluation
Queries

- **Performance Summary**

	Operation	VV	VE	VF	FV	FE	FF	EV	EF
Order	default	4.95	3.48	4.8	1.27	1.05	0.87	0.86	0.64
	sorted	3.92	2.89	3.77	1.04	0.93	0.72	0.86	0.63
	shuffle	8.37	5.48	8.19	3.86	2.01	2.55	0.85	0.62

RXMesh speedup over PDE

Evaluation
Queries

- **Performance Summary**
  - Using shared memory to capture locality is more effective for these operations

Order	Operation	VV	VE	VF	FV	FE	FF	EV	EF
	default	4.95	3.48	4.8	1.27	1.05	0.87	0.86	0.64
	sorted	3.92	2.89	3.77	1.04	0.93	0.72	0.86	0.63
	shuffle	8.37	5.48	8.19	3.86	2.01	2.55	0.85	0.62

RXMesh speedup over PDE

Evaluation
Queries

- **Performance Summary**
  - PDE only writes two (4-byte) numbers per thread for these operations and thus it is ~1.6X faster

	Operation	VV	VE	VF	FV	FE	FF	EV	EF
Order	default	4.95	3.48	4.8	1.27	1.05	0.87	0.86	0.64
	sorted	3.92	2.89	3.77	1.04	0.93	0.72	0.86	0.63
	shuffle	8.37	5.48	8.19	3.86	2.01	2.55	0.85	0.62

RXMesh speedup over PDE

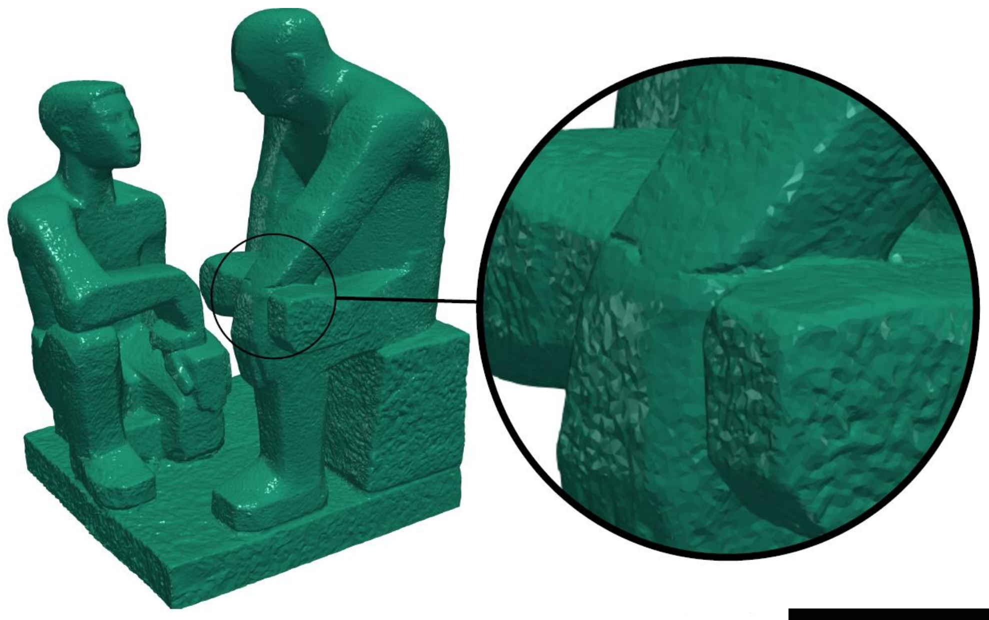
Evaluation
Applications

- **Mean Curvature Flow** [Desbrun et al. 1991]
  - Using matrix-free conjugate gradient solver

Evaluation
Applications

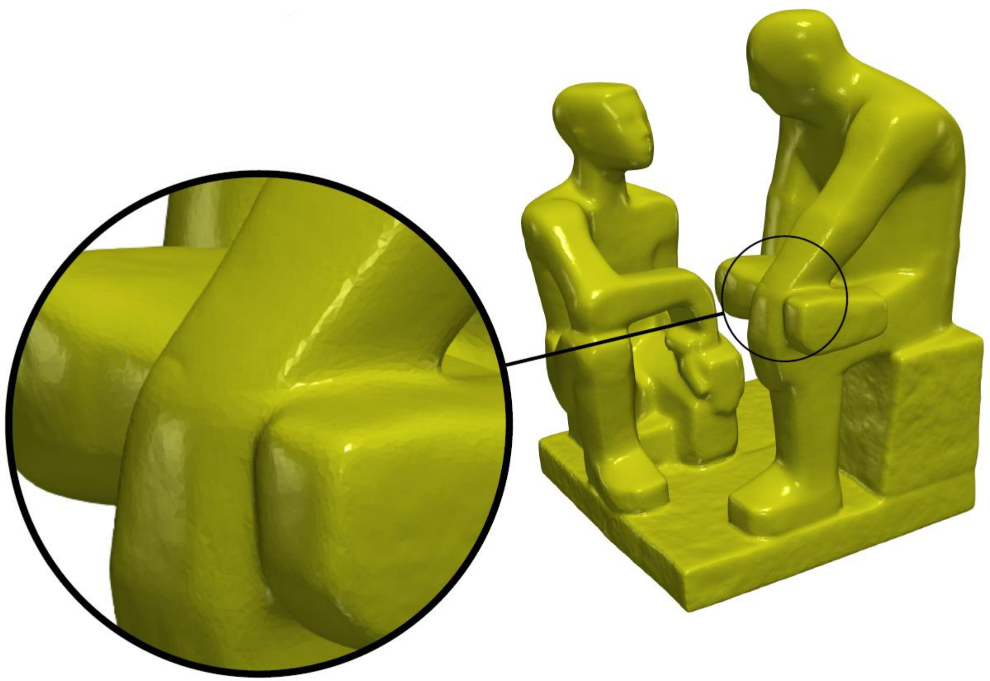
SIGGRAPH 2021

- **Mean Curvature Flow** [Desbrun et al. 1991]
  - Using matrix-free conjugate gradient solver



Input

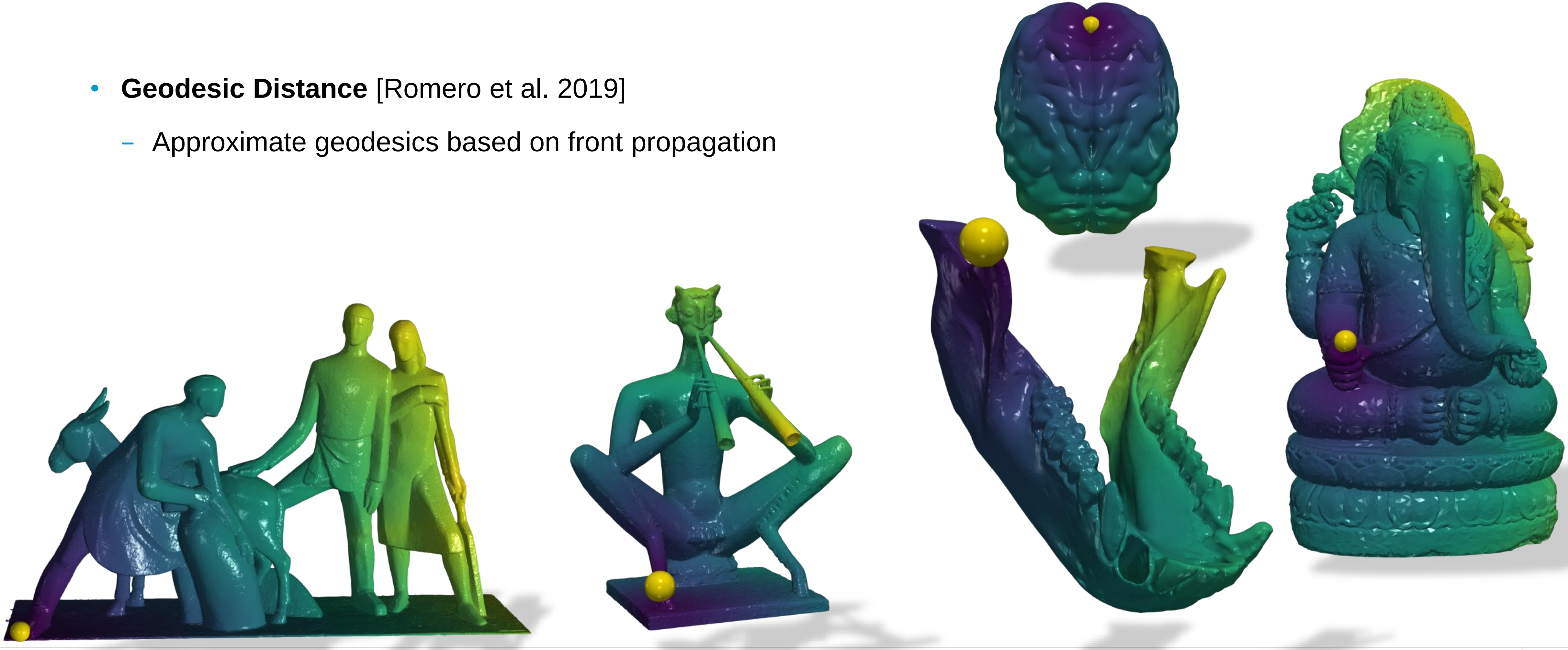
RXMesh **4.6x** faster than PDE



Output

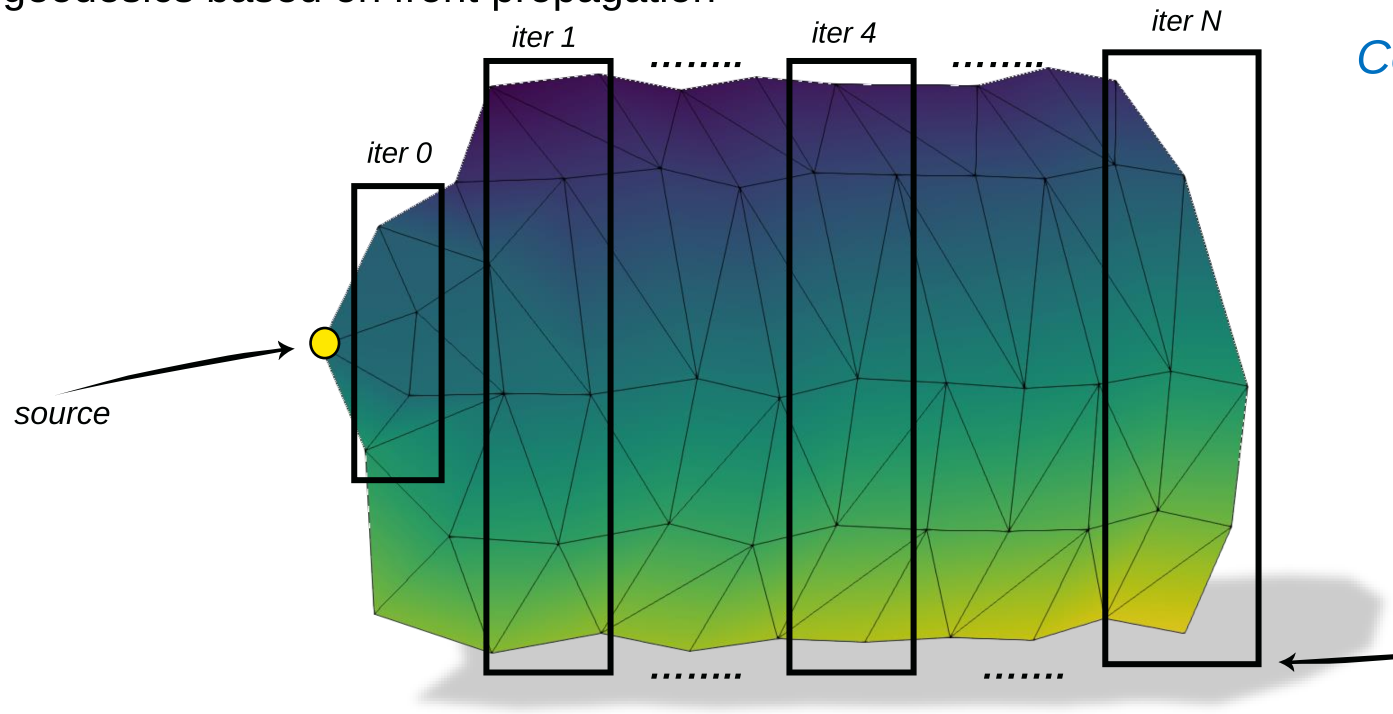
Evaluation
Applications

- **Geodesic Distance** [Romero et al. 2019]
  - Approximate geodesics based on front propagation



Evaluation
Applications

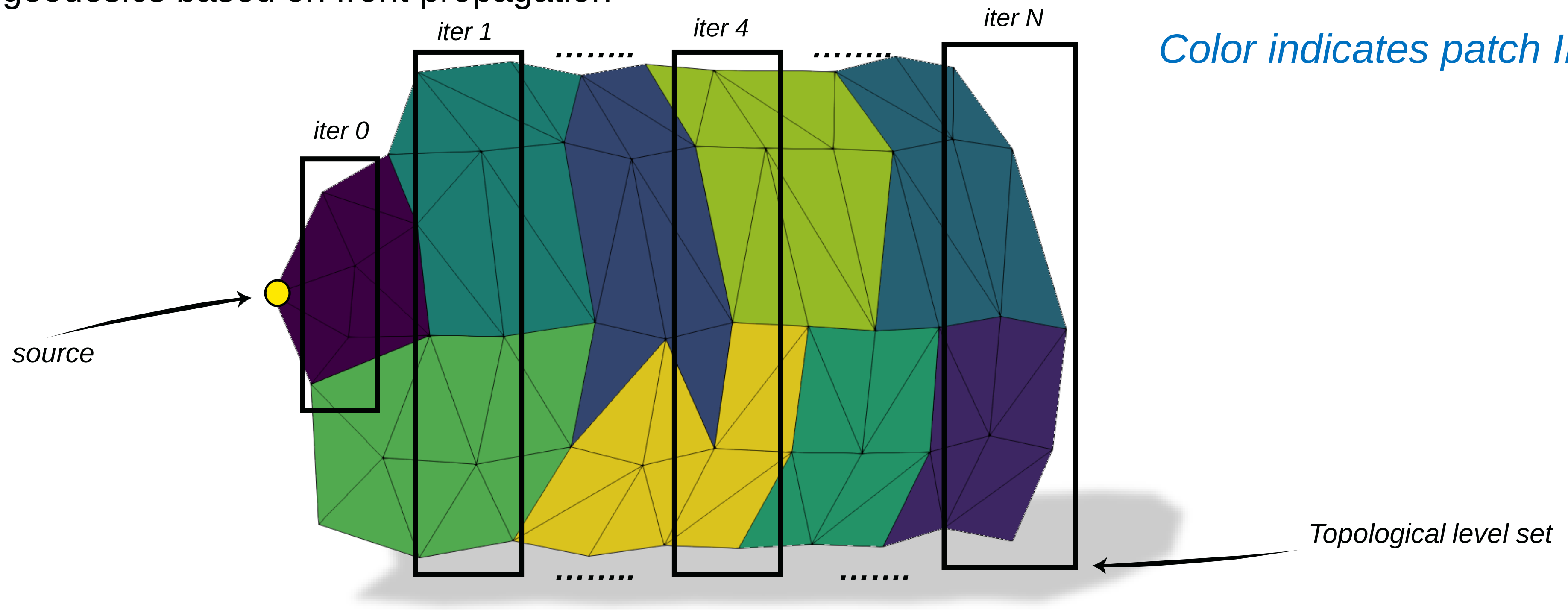
- **Geodesic Distance** [Romero et al. 2019]
  - Approximate geodesics based on front propagation



Color indicates face index

Evaluation
Applications

- **Geodesic Distance** [Romero et al. 2019]
  - Approximate geodesics based on front propagation



Color indicates patch ID

Evaluation
Applications

- **Geodesic Distance** [Romero et al. 2019]
  - Approximate geodesics based on front propagation

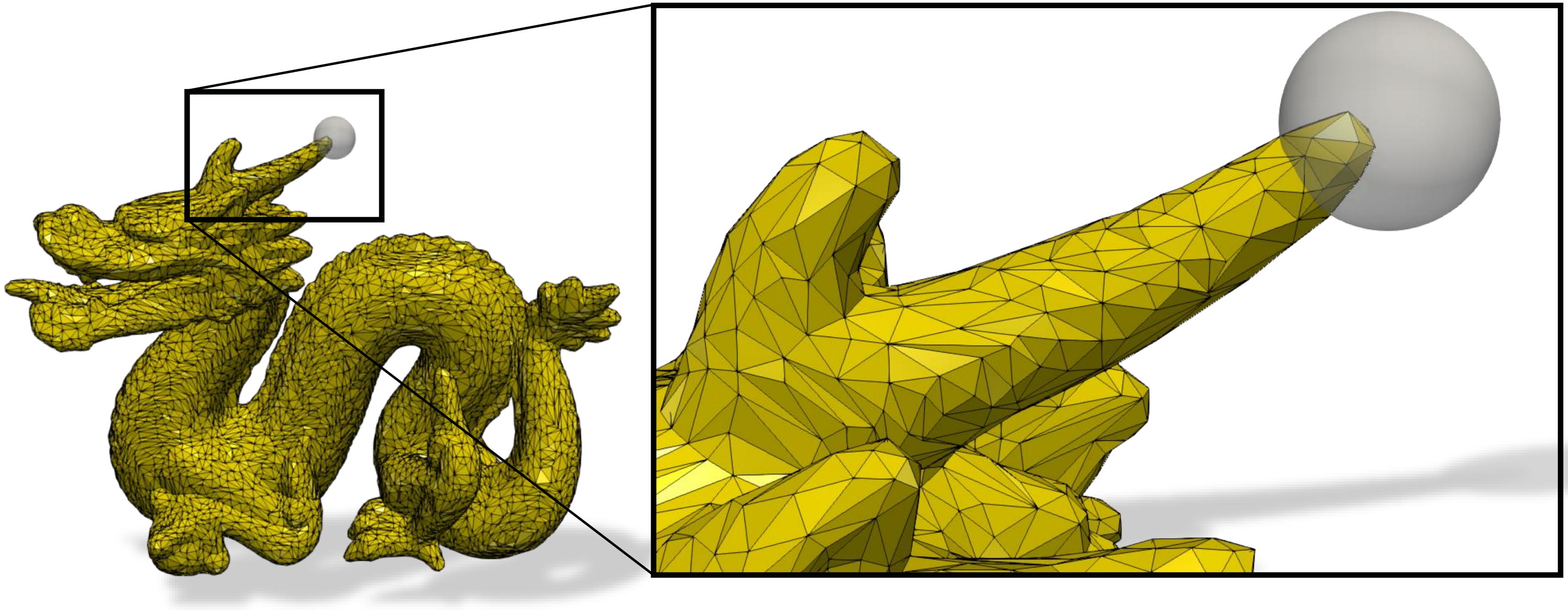
RXMesh **15.5x** faster than PDE

Evaluation
Applications

- **Bilateral Filtering** [Fleishman et al. 2003]
  - Explores RXMesh's performance to generate *k*-ring queries

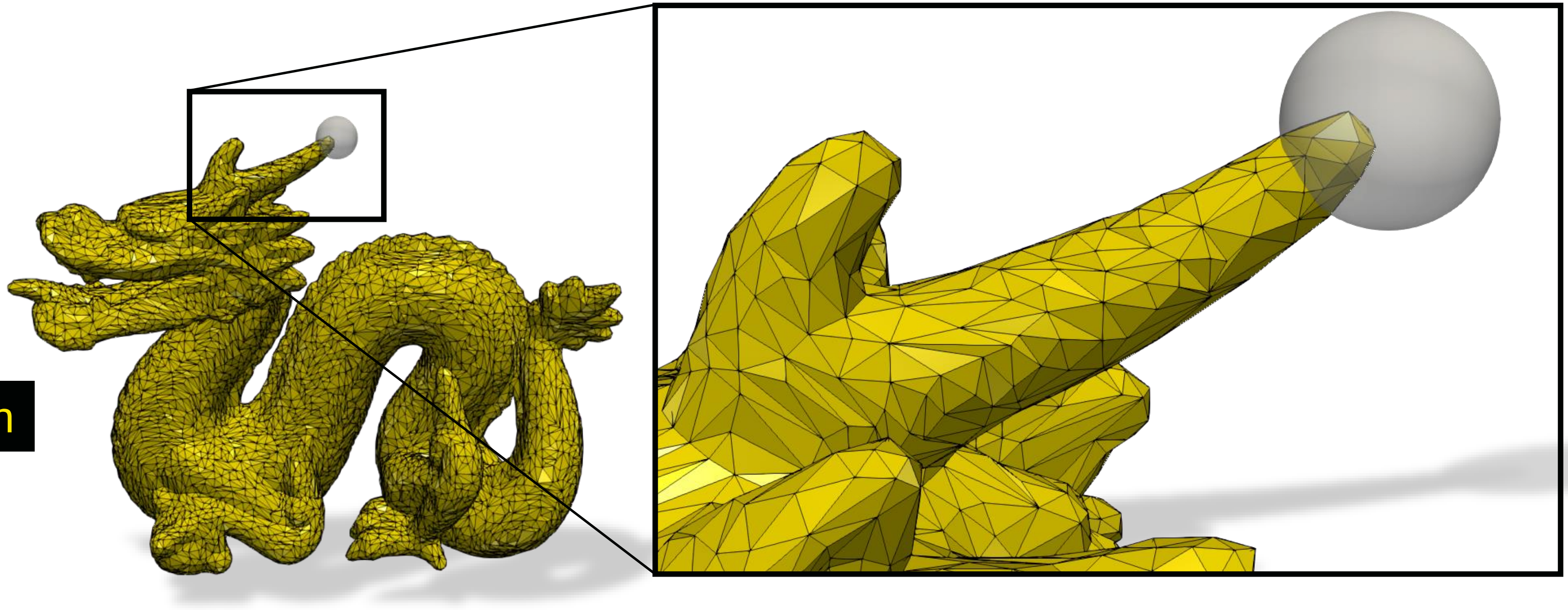
Evaluation
Applications

- **Bilateral Filtering** [Fleishman et al. 2003]
  - Explores RXMesh's performance to generate *k*-ring queries



Evaluation
Applications

- **Bilateral Filtering** [Fleishman et al. 2003]
  - Explores RXMesh's performance to generate *k*-ring queries



PDE is **1.12x** faster than RXMesh

Evaluation
Applications

- **Vertex Normal** [Max 1999]
  - Compares RXMesh's performance against hard-wired data structure i.e., indexed triangles

Evaluation
Applications

- **Vertex Normal** [Max 1999]
  - Compares RXMesh's performance against hard-wired data structure.

Indexed triangle is **1.14x** faster than RXMesh

Future work

SIGGRAPH 2021

- Support for dynamic changes
  - What are the right semantics?

Future work

SIGGRAPH 2021

- Support for dynamic changes
  - What are the right semantics?
- Improve higher-order queries' performance
- Extension to quad mesh
  - and maybe volumetric mesh (?)

Conclusion

SIGGRAPH 2021

Programmer-managed caching is the right way to capture mesh locality and improve GPU performance for mesh processing

RXMesh: A GPU Mesh Data Structure

Github.com/OwensGroup/RXMesh

ahmahmoud@ucdavis.edu

Funded by: NSF, NVIDIA, DARPA, Sandia National Laboratories

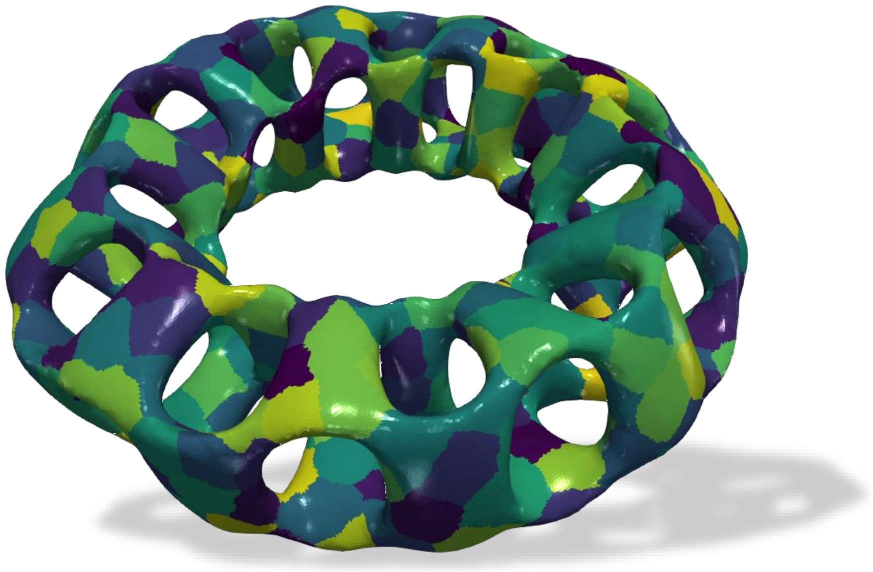
Backup Slides

Design Principles

SIGGRAPH 2021

1. Locality by Patching

- Patch quality:
  - Small patches (~512-768 faces/patch)
  - Contiguous i.e., each patch is a single component
  - As equal-sized as possible



Design Principles

SIGGRAPH 2021

1. Locality by Patching

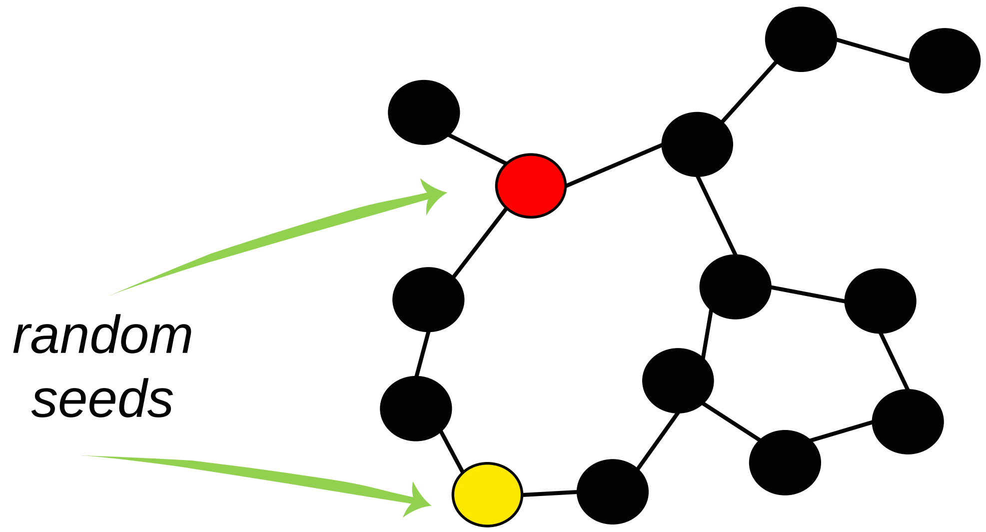
- Patching algorithm:
    - Inspired by Lloyd’s clustering algorithm for graphs

Design Principles

SIGGRAPH 2021

1. Locality by Patching

- Patching algorithm:
  - Inspired by Lloyd's clustering algorithm for graphs
    - Step 0: random seeds

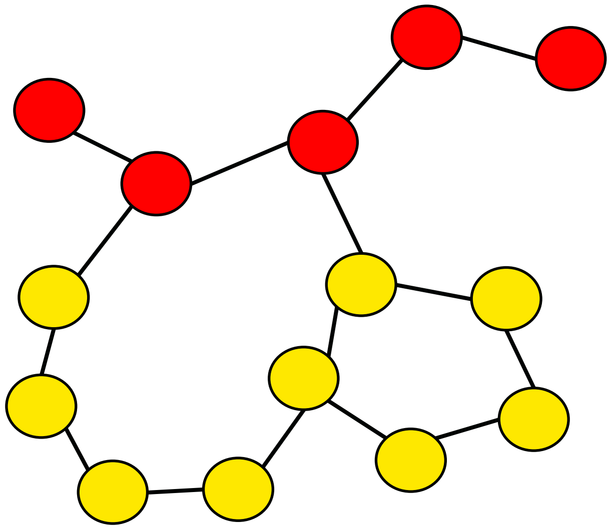


Design Principles

SIGGRAPH 2021

1. Locality by Patching

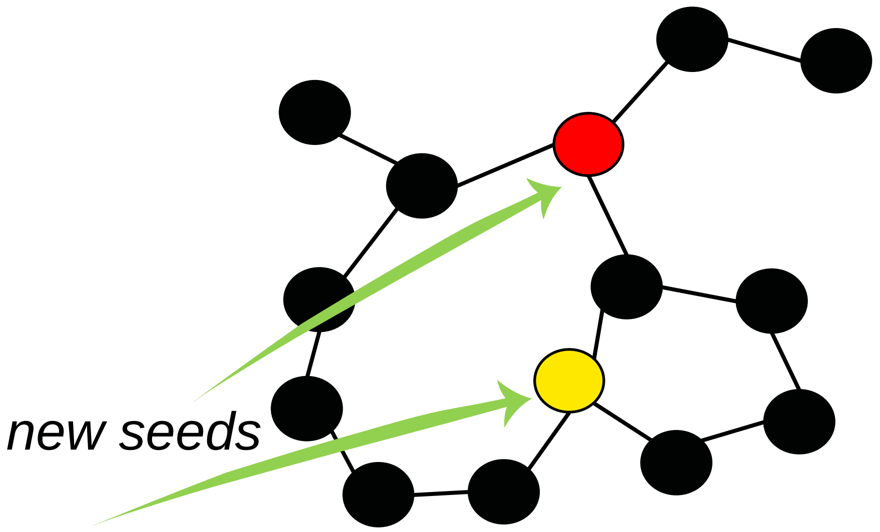
- Patching algorithm:
  - Inspired by Lloyd's clustering algorithm for graphs
    - Step 0: random seeds
    - While (not converged)
      - Step 1: assign vertices to nearest seed



Design Principles

1. Locality by Patching

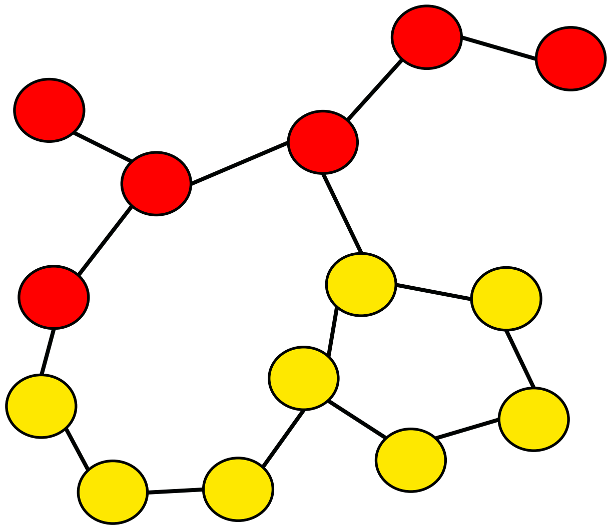
- Patching algorithm:
  - Inspired by Lloyd’s clustering algorithm for graphs
    - Step 0: random seeds
    - While (not converged)
      - Step 1: assign vertices to nearest seed
      - Step 2: update petition's seed with its centroid



Design Principles

1. Locality by Patching

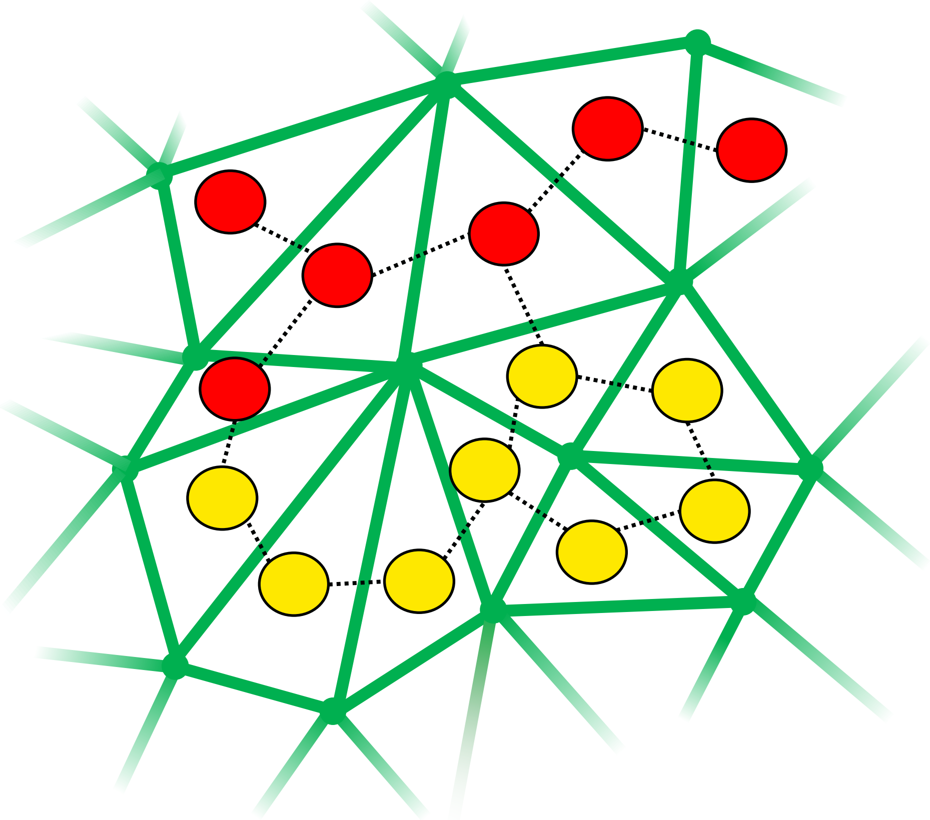
- Patching algorithm:
    - Inspired by Lloyd’s clustering algorithm for graphs
        - Step 0: random seeds
        - While (not converged)
            - Step 1: assign vertices to nearest seed
            - Step 2: update petition's seed with its centroid



Design Principles

1. Locality by Patching

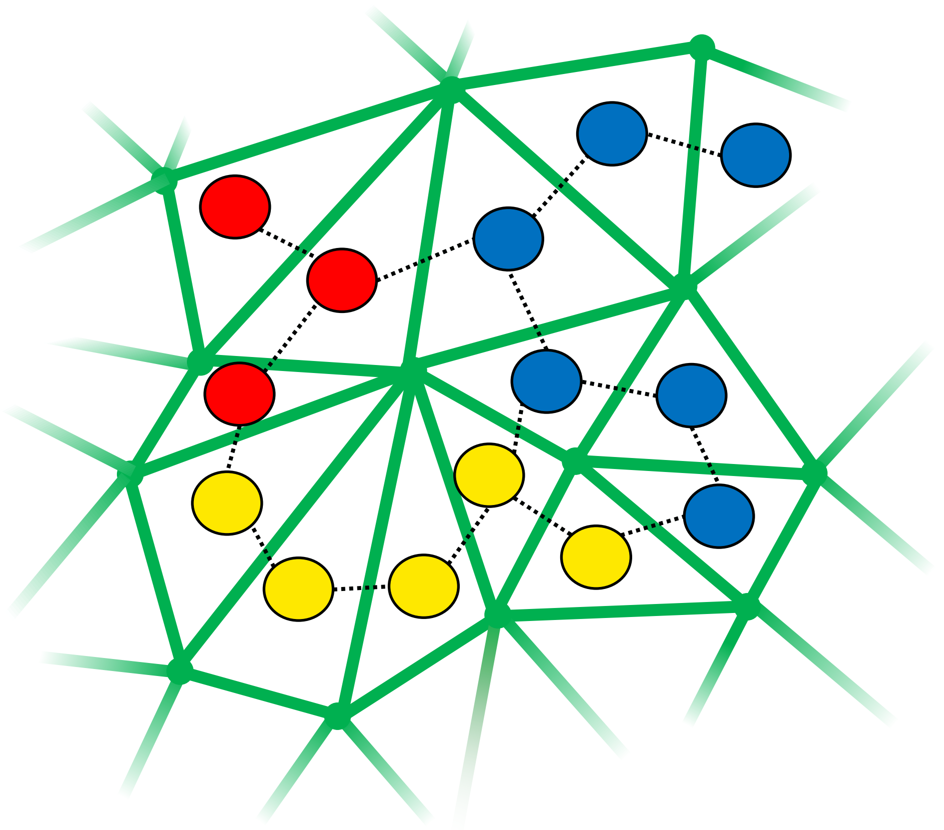
- Patching algorithm:
  - Inspired by Lloyd's clustering algorithm for graphs
    - Step 0: random seeds
    - While (not converged)
      - Step 1: assign vertices to nearest seed
      - Step 2: update petition's seed with its centroid



Design Principles

1. Locality by Patching

- Patching algorithm:
  - Inspired by Lloyd's clustering algorithm for graphs
    - Step 0: random seeds
    - While (not converged)
      - Step 1: assign vertices to nearest seed
      - Step 2: update petition's seed with its centroid
      - Step 3: add addition seeds every 5th iteration

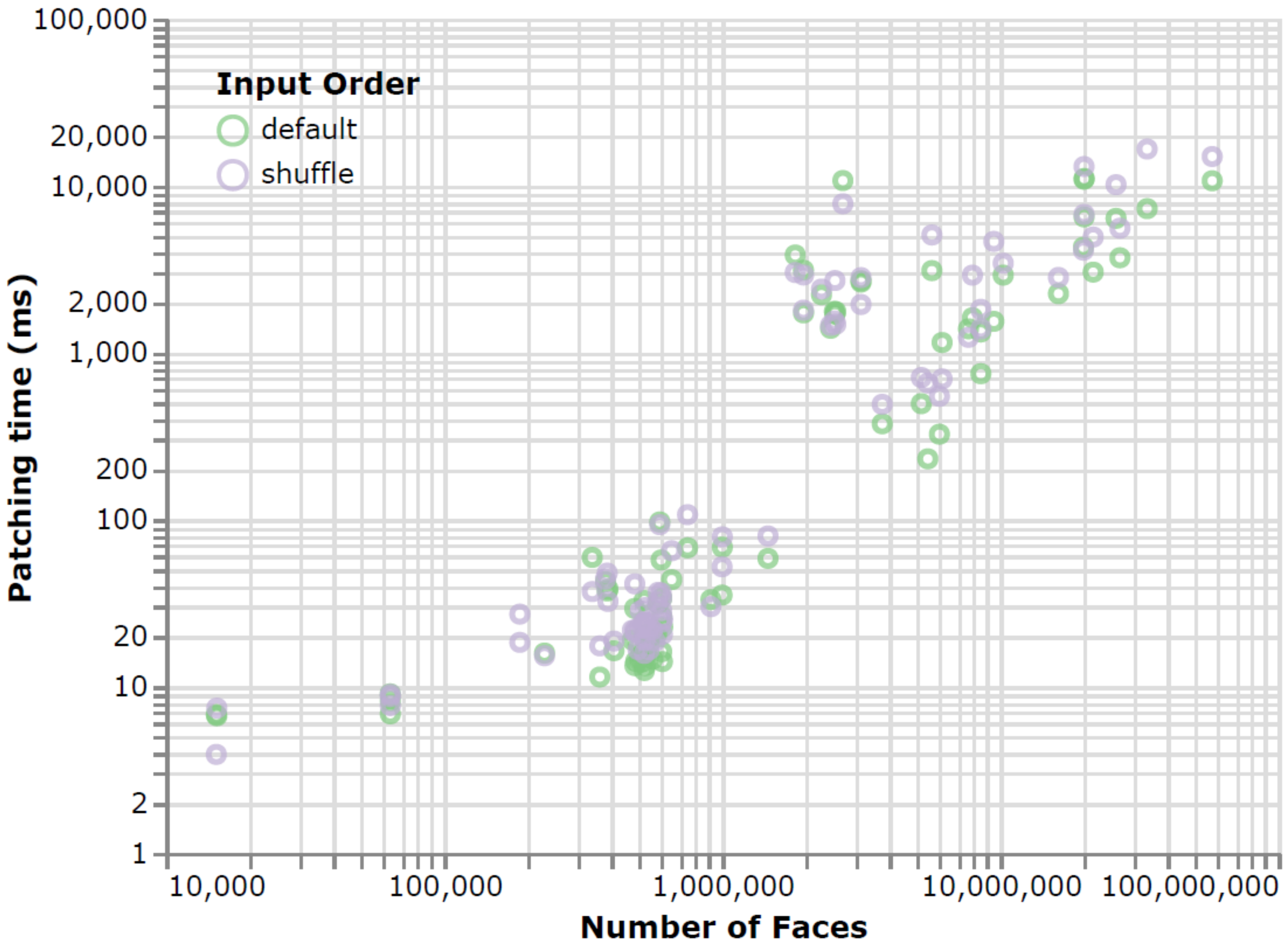


Design Principles

SIGGRAPH 2021

1. **Locality by Patching**

Less than *100 ms* for *1M* faces

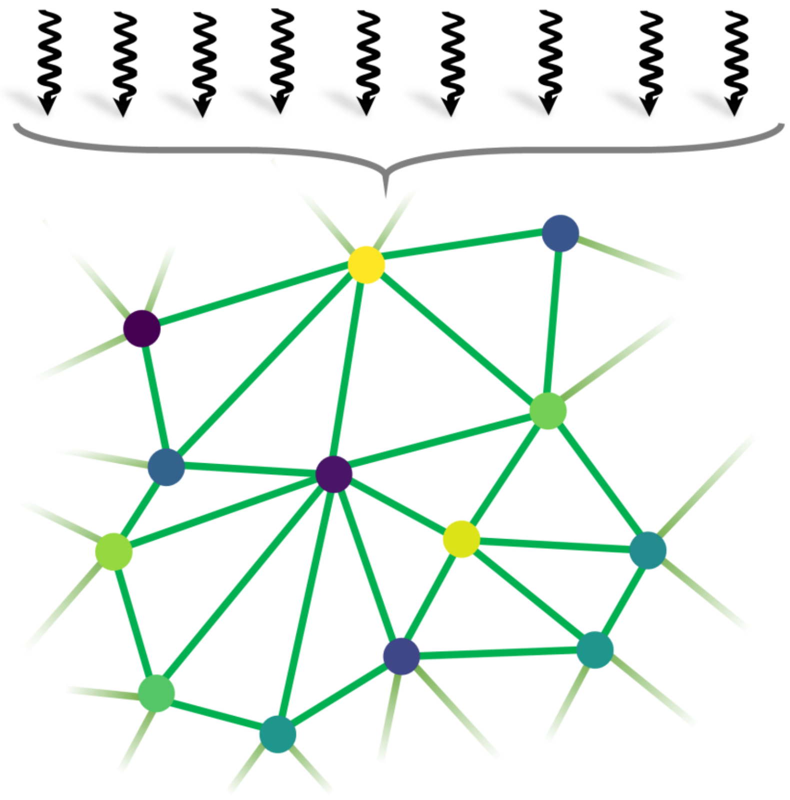


Query Pipeline

- **Every block does:**
  - Reads a patch from global memory into shared memory

Query Pipeline

- **Every block does:**
  - Reads a patch from global memory into shared memory
  - Performs the respective query



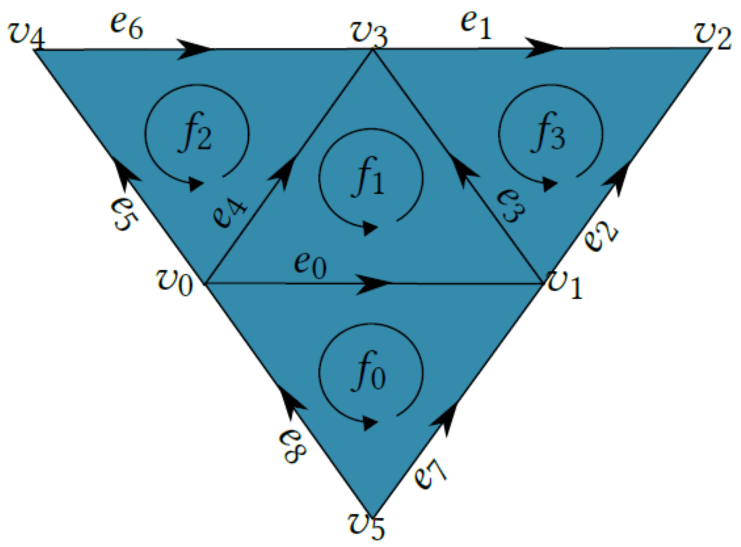
Threads work ***collaboratively***

Query Pipeline

SIGGRAPH 2021

- **Every block does:**
  - Reads a patch from global memory into shared memory
  - Performs the respective query
  - Maps the output query into global index space

Memory Layout



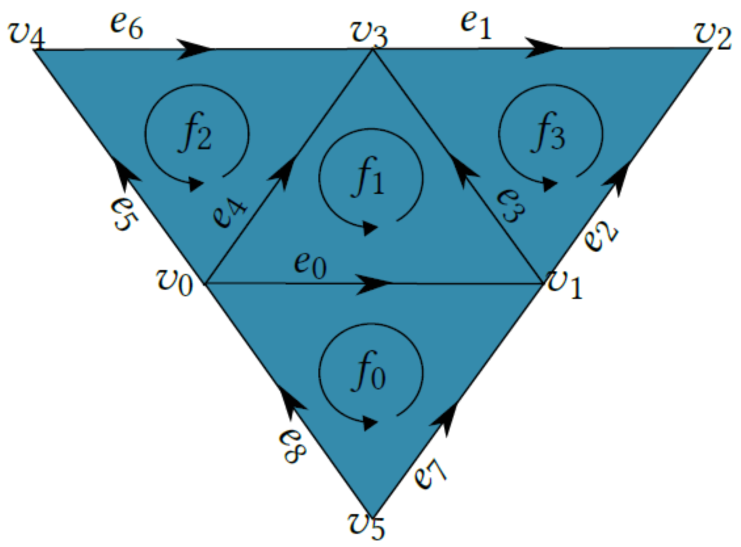
	v_0	v_1	v_2	v_3	v_4	v_5
e_0	-1	1				
e_1			1	-1		
e_2		-1	1			
e_3		-1		1		
e_4	-1			1		
e_5	-1				1	
e_6				1	-1	
e_7		1				-1
e_8	1					-1

M_{EV}

$$\left[\underbrace{0 \quad 1}_{e_0} \,\Big|\, \underbrace{3 \quad 2}_{e_1} \,\Big|\, \underbrace{1 \quad 2}_{e_2} \,\Big|\, \underbrace{1 \quad 3}_{e_3} \,\Big|\, \underbrace{0 \quad 3}_{e_4} \,\Big|\, \underbrace{0 \quad 4}_{e_5} \,\Big|\, \underbrace{4 \quad 3}_{e_6} \,\Big|\, \underbrace{5 \quad 1}_{e_7} \,\Big|\, \underbrace{5 \quad 0}_{e_8}\right]$$

Compact M_{EV}

Memory Layout



	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
f_0	-2							1	-3
f_1	1			2	-3				
f_2					1	-3	-2		
f_3		-2	1	-3					

M_{FE}

$$\left[\; \underbrace{7 \quad -0 \quad -8}_{f_0} \;\middle|\; \underbrace{0 \quad 3 \quad -4}_{f_1} \;\middle|\; \underbrace{4 \quad -6 \quad -5}_{f_2} \;\middle|\; \underbrace{2 \quad -1 \quad -3}_{f_3} \;\right]$$

Compact M_{FE}