RXMesh: A GPU Mesh Data Structure

Ahmed H. Mahmoud\textsuperscript{1,2}, Serban D. Porumbescu\textsuperscript{1}, and John D. Owens\textsuperscript{1}

\textsuperscript{1}University of California, Davis; \textsuperscript{2}Autodesk Research
Motivation

- Triangle meshes are everywhere

Boundary First Flattening [Sawhney et al., 2018]

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

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

- 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
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

- 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

- 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
- Vertex normal computation
RXMesh Programming Model

- Vertex normal computation
RXMesh Programming Model

- 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
Vertex normal computation

```c
__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);});
```
Vertex normal computation

```c
__global__ void
ComputeVertexNormal(RXMesh rxmesh,
                     Vec3<float>* VertexNormals,
                     const Vec3<float>* VertexCoords) {

  rxmesh.template<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);
                          });
  }
```
• Vertex normal computation

```c
__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);
        });
}
```
Vertex normal computation

```cpp
__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);
        });
}
```
• Vertex normal computation

```c
__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

**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
RXMesh Data Structure:
- Design Goals
- Design Principles
## What queries RXMesh supports

<table>
<thead>
<tr>
<th>Query</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>VV</td>
<td>For vertex V, return adjacent vertices</td>
</tr>
<tr>
<td>VE</td>
<td>For vertex V, return incident edges</td>
</tr>
<tr>
<td>VF</td>
<td>For vertex V, return incident faces</td>
</tr>
<tr>
<td>EV</td>
<td>For edge E, return incident vertices</td>
</tr>
<tr>
<td>EF</td>
<td>For edge E, return incident faces</td>
</tr>
<tr>
<td>FV</td>
<td>For face F, return incident vertices</td>
</tr>
<tr>
<td>FE</td>
<td>For face F, return incident edges</td>
</tr>
<tr>
<td>FF</td>
<td>For face F, return adjacent faces</td>
</tr>
</tbody>
</table>
1. Performance
   - Improve locality and confine computation within the shared memory
Design Goals

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

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
1. Locality by Patching

Global Sorting

Color indicates face index
1. Locality by Patching

Global Sorting
Color indicates face index

Patching
Color indicates patch ID
2. Patch Representation

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

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

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

![Diagram of a triangle with vertices labeled and arrows indicating edges.]

<table>
<thead>
<tr>
<th></th>
<th>(v_0)</th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
<th>(v_4)</th>
<th>(v_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_0)</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e_1)</td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>(e_2)</td>
<td>-1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(e_3)</td>
<td></td>
<td>-1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(e_4)</td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>(e_5)</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>(e_6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>(e_7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>(e_8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\[M_{EV}\]
2. Patch Representation

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

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

Threads work *independently*
3. Work Mapping

Threads work collaboratively

CUDA Thread
4. Ribbons

- Patch-boundary mesh elements require special treatment
4. Ribbons

- Patch-boundary mesh elements require special treatment
5. Index Spaces

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

\[
M_{L6} = 
\begin{bmatrix}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
M_{G6} = 
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
Evaluation:
1. Query Operations
2. Applications
   - Mean Curvature Flow
   - Geodesic Distance
   - Bilateral Filtering
   - Vertex Normal
Evaluation

- 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

- Input order

Color indicates face index
Evaluation

- Input order

Color indicates face index

Sorted

Default
Evaluation

- Input order

Shuffled

Sorted

Default

Color indicates face index
### Evaluation Queries

- **Performance Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>VV</th>
<th>VE</th>
<th>VF</th>
<th>FV</th>
<th>FE</th>
<th>FF</th>
<th>EV</th>
<th>EF</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>4.95</td>
<td>3.48</td>
<td>4.8</td>
<td>1.27</td>
<td>1.05</td>
<td>0.87</td>
<td>0.86</td>
<td>0.64</td>
</tr>
<tr>
<td>sorted</td>
<td>3.92</td>
<td>2.89</td>
<td>3.77</td>
<td>1.04</td>
<td>0.93</td>
<td>0.72</td>
<td>0.86</td>
<td>0.63</td>
</tr>
<tr>
<td>shuffle</td>
<td>8.37</td>
<td>5.48</td>
<td>8.19</td>
<td>3.86</td>
<td>2.01</td>
<td>2.55</td>
<td>0.85</td>
<td>0.62</td>
</tr>
</tbody>
</table>

**RXMesh speedup over PDE**
### Evaluation Queries

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>VV</th>
<th>VE</th>
<th>VF</th>
<th>FV</th>
<th>FE</th>
<th>FF</th>
<th>EV</th>
<th>EF</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>4.95</td>
<td>3.48</td>
<td>4.8</td>
<td>1.27</td>
<td>1.05</td>
<td>0.87</td>
<td>0.86</td>
<td>0.64</td>
</tr>
<tr>
<td>sorted</td>
<td>3.92</td>
<td>2.89</td>
<td>3.77</td>
<td>1.04</td>
<td>0.93</td>
<td>0.72</td>
<td>0.86</td>
<td>0.63</td>
</tr>
<tr>
<td>shuffle</td>
<td>8.37</td>
<td>5.48</td>
<td>8.19</td>
<td>3.86</td>
<td>2.01</td>
<td>2.55</td>
<td>0.85</td>
<td>0.62</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>VV</th>
<th>VE</th>
<th>VF</th>
<th>FV</th>
<th>FE</th>
<th>FF</th>
<th>EV</th>
<th>EF</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>4.95</td>
<td>3.48</td>
<td>4.8</td>
<td>1.27</td>
<td>1.05</td>
<td>0.87</td>
<td>0.86</td>
<td>0.64</td>
</tr>
<tr>
<td>sorted</td>
<td>3.92</td>
<td>2.89</td>
<td>3.77</td>
<td>1.04</td>
<td>0.93</td>
<td>0.72</td>
<td>0.86</td>
<td>0.63</td>
</tr>
<tr>
<td>shuffle</td>
<td>8.37</td>
<td>5.48</td>
<td>8.19</td>
<td>3.86</td>
<td>2.01</td>
<td>2.55</td>
<td>0.85</td>
<td>0.62</td>
</tr>
</tbody>
</table>

RXMesh speedup over PDE
• **Mean Curvature Flow** [Desbrun et al. 1991]
  - Using matrix-free conjugate gradient solver
Evaluation
Applications

- **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

Topological level set

© 2021 SIGGRAPH. ALL RIGHTS RESERVED.
Evaluation
Applications

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

![Diagram showing a topological level set with iterations labeled iter 0, iter 1, iter 4, and iter N. Color indicates patch ID.](image)

Topological level set

Color indicates patch ID
Evaluation
Applications

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

RXMesh 15.5x faster than PDE
• **Bilateral Filtering** [Fleishman et al. 2003]
  - Explores RXMesh’s performance to generate $k$-ring queries
• **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

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

• Support for dynamic changes
  – What are the right semantics?

• Improve higher-order queries’ performance

• Extension to quad mesh
  – and maybe volumetric mesh (?)
Conclusion

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
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
1. Locality by Patching

- Patching algorithm:
  - Inspired by Lloyd's clustering algorithm for graphs
1. Locality by Patching

- Patching algorithm:
  - Inspired by Lloyd’s clustering algorithm for graphs
    - Step 0: random seeds
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
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
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
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
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
1. Locality by Patching

Less than 100 ms for 1M faces
• 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
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

The diagram shows a triangle with vertices labeled $v_0$, $v_1$, $v_2$, $v_3$, $v_4$, and $v_5$, and edges $e_0$, $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$, and $e_7$. The edges are labeled with the labels $f_0$, $f_1$, and $f_3$.

A matrix $M_{EV}$ is shown:

$$
M_{EV} = \begin{bmatrix}
0 & 1 & 3 & 2 & 1 & 2 & 1 & 3 & 0 & 3 & 0 & 4 & 4 & 3 & 5 & 1 & 5 & 0 \\
\{e_0\} & \{e_1\} & \{e_2\} & \{e_3\} & \{e_4\} & \{e_5\} & \{e_6\} & \{e_7\} & \{e_8\}
\end{bmatrix}
$$

The compact form of $M_{EV}$ is:

$$
\begin{bmatrix}
0 & 1 & 3 & 2 & 1 & 2 & 1 & 3 & 0 & 3 & 0 & 4 & 4 & 3 & 5 & 1 & 5 & 0 \\
\end{bmatrix}
$$
Memory Layout

\[
M_{FE} = \begin{bmatrix}
7 & -0 & -8 \\
0 & 3 & -4 \\
4 & -6 & -5 \\
\end{bmatrix}
\]

Compact \( M_{FE} \)

\[
\begin{array}{c|c|c|c|c|c|c|c}
& e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \\
\hline
f_0 & -2 & 1 & 2 & 2 & -3 & -3 & 1 & -3 \\
f_1 & 1 & 2 & 1 & 1 & -2 & 1 & -3 & 1 \\
f_2 & -2 & 1 & -3 & 1 & -3 & 1 & -3 & 1 \\
f_3 & 1 & -3 & 1 & -3 & 1 & -3 & 1 & -3 \\
\end{array}
\]