EECE5698
Networked XR Systems

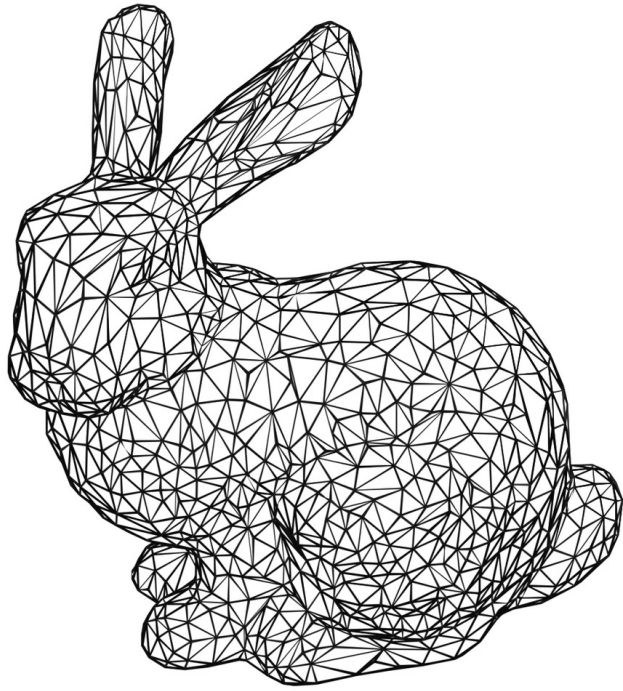
Lecture Outline for Today

- Mesh Compression

Recap: Mesh

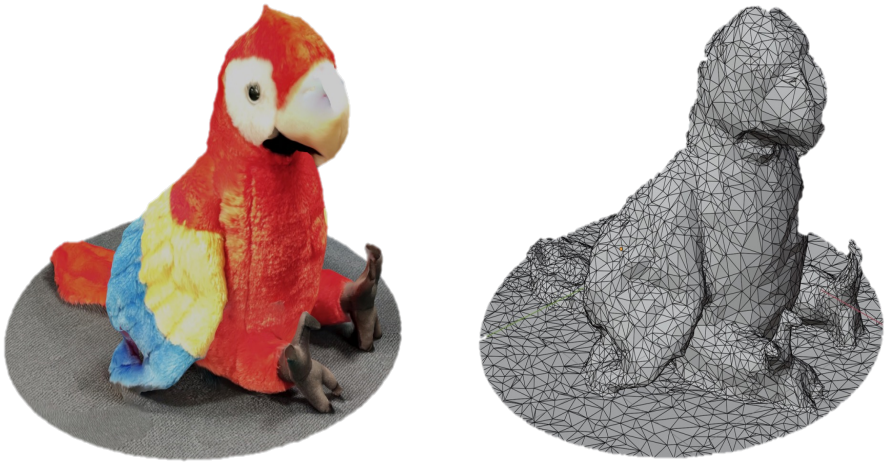
- A set of polygons, connected by their common edges or vertices
- Typically represented by triangles

- Meshes are fundamental to rendering scenes in video games, animations, XR, and more.



Recap: Mesh

- Data representation
  - Each frame has vertices and connectivity
  - Color texture is stored independently, so there is also mapping information from texture to polygons



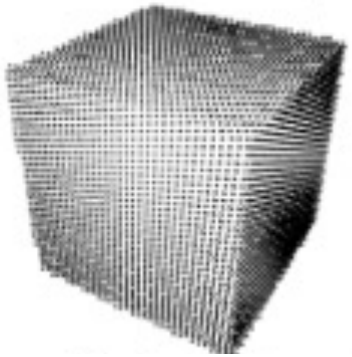
Why Mesh Compression

- **Challenges**: Large meshes consume significant memory and bandwidth, making storage and transmission inefficient.
- **Objectives**: Compress meshes to reduce file size without significantly losing quality, enabling faster loading times and lower storage requirements.
- **Benefits**: Efficient mesh compression improves performance in real-time applications and reduces costs in data transmission and storage.

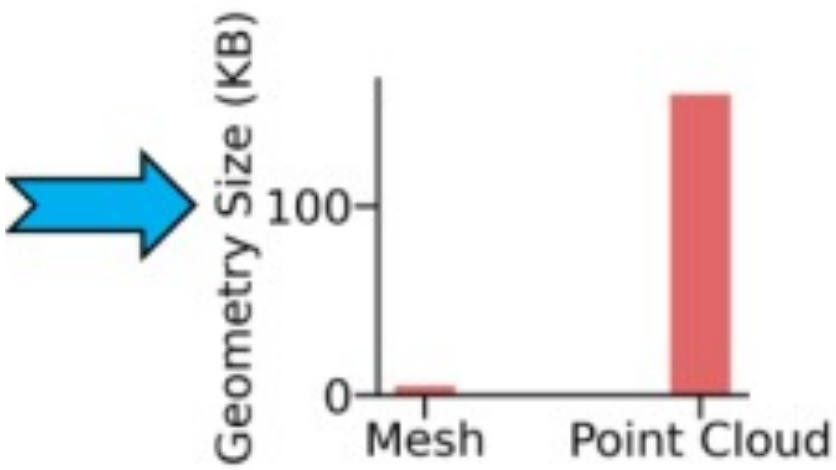
Mesh vs. Point Cloud

- Meshes are much more compact

Point cloud



Triangle mesh



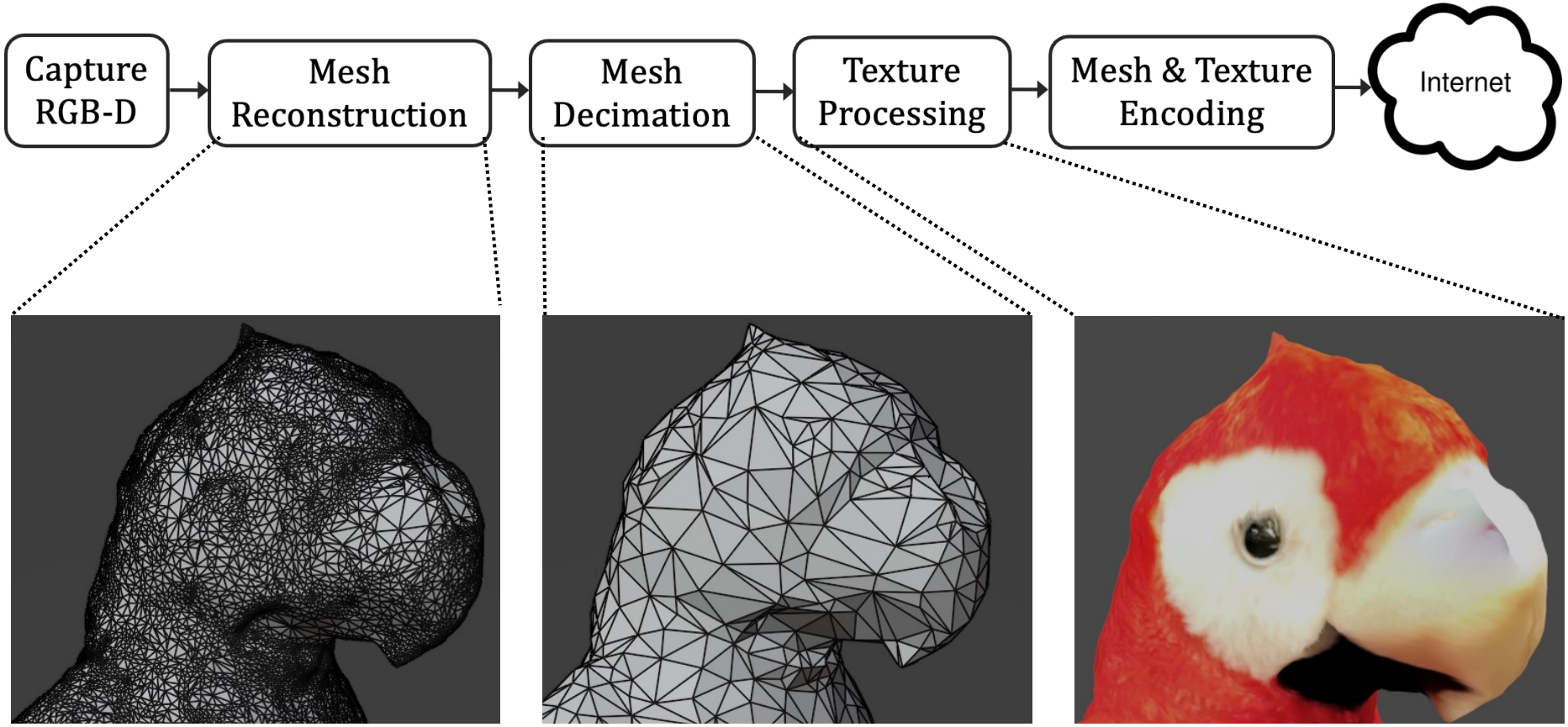
Counter Intuitive from the previous slide?

Mesh Compression

- Mesh Simplification – Vertex clustering or quadratic error decimation
- Vertex Compression
- Connectivity Compression
- Texture Compression

Mesh Compression

Mesh simplification can be a form of compression



Mesh Compression

- Vertex Compression
  - Reduce the size of vertex coordinates while preserving the mesh's geometric detail.
- Techniques
  - **Quantization**: Converts floating-point coordinates to a fixed number of bits, reducing precision but saving space.
  - **Predictive Coding**: Encodes vertex positions as differences from predicted positions based on previous vertices, exploiting spatial coherence.
- Example: Using delta encoding, where each vertex position is stored as the difference from the previous vertex, significantly reducing the range of values.

Mesh Compression

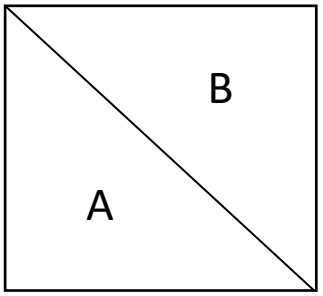
- Vertex Compression
  - Reduce the size of vertex coordinates while preserving the mesh's geometric detail.
- Techniques:
  - Vertices can be considered same as point cloud
  - **Can we use point cloud compression techniques that we discussed in the previous lecture**?

Mesh Compression

- Connectivity Compression
  - Efficiently encoding the mesh topology, which defines how vertices are connected to form faces.
- Techniques – Edgebreaker algorithm
  - The algorithm traverses the mesh, encoding its topology with a sequence of symbols representing the traversal operations.
  - Includes symbols like C (connect), L (left), R (right), E (end), and S (start), which describe how to move from one triangle to the next – CLERS String

Mesh Compression

- Edgebreaker Algorithm
  - The algorithm starts at an edge of the mesh and follows the edges around the mesh in a systematic way, essentially "breaking" the edges as it goes to avoid retracing its path. This traversal forms a loop around the mesh, visiting each triangle once.



Step 1: Start at the outer edge of **A** (S could denote this start, but it's optional).
Step 2: Move to triangle **B** via the shared edge — since **B** is directly connected without requiring a turn, this move is encoded as "C" (Connect).
Step 3: From **B**, there's no new triangle to visit, so the algorithm would end — this could be marked with "E" for End, but since it's a simple case, the end might be implicit.

Mesh Compression

- Connectivity Compression
  - Efficiently encoding the mesh topology, which defines how vertices are connected to form faces.
- Techniques – Edgebreaker algorithm
  - Achieves at most 4 bits per vertex
  - Published in 1996 but popular even today
  - Used in Google's Draco Mesh compression code (https://google.github.io/draco)

Mesh Compression

- Connectivity Compression
  - Efficiently encoding the mesh topology, which defines how vertices are connected to form faces.
- Techniques – Edgebreaker algorithm
  - Limitations: the algorithm assumes a manifold mesh, which may limit its applicability to meshes with more complex topologies without preprocessing

Definition of manifold mesh: if you were a tiny ant walking on the surface of the 3D model, you could walk all over the model without ever finding a place where the surface doesn't make sense — no holes, no edges hanging in the air, and no overlapping faces.

Mesh Compression

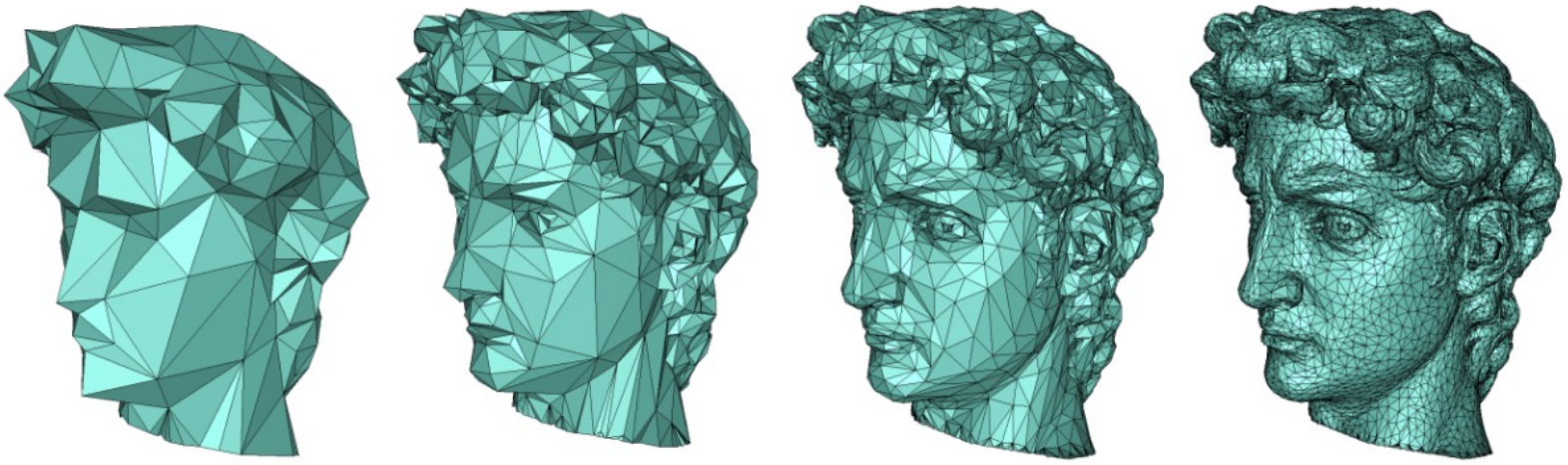
- Connectivity Compression
  - Efficiently encoding the mesh topology, which defines how vertices are connected to form faces.
- Techniques
  - Edgebreaker – lossy for non-manifold meshes
  - TFAN (Triangle fan) algorithm – lossless
  - Valence-driven encoding – based on the number of connected edges

Mesh Compression

- Texture compression
  - How?

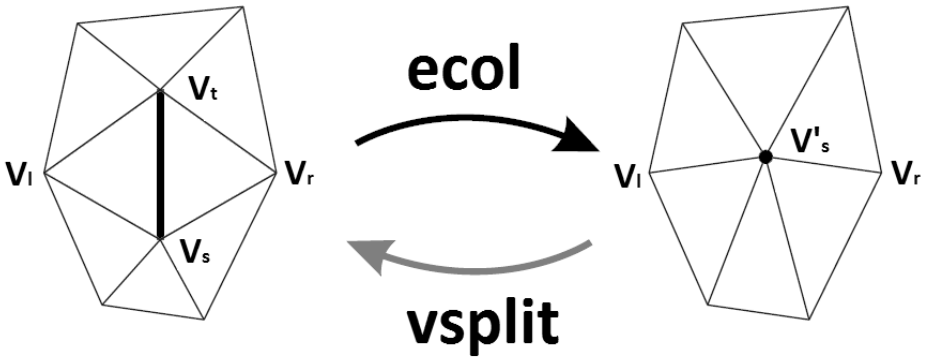
Mesh Compression

- Progressive compression
  - Different levels of detail are created by simplifying the original mesh step by step, usually by vertex decimation or edge collapse techniques.



Mesh Compression

- Progressive compression
  1. **Edge Collapse**: In the simplification process, an operation called "edge collapse" is frequently used, where an edge between two vertices is collapsed into a single vertex, reducing the overall count of vertices and faces.
  2. **Vertex Split**: The reverse of edge collapse is "vertex split." To refine the mesh, the algorithm splits a vertex into two and recreates the original edge and associated faces. The vertex split operation is stored as a record of how to refine the mesh from one LOD to the next.



Mesh Compression

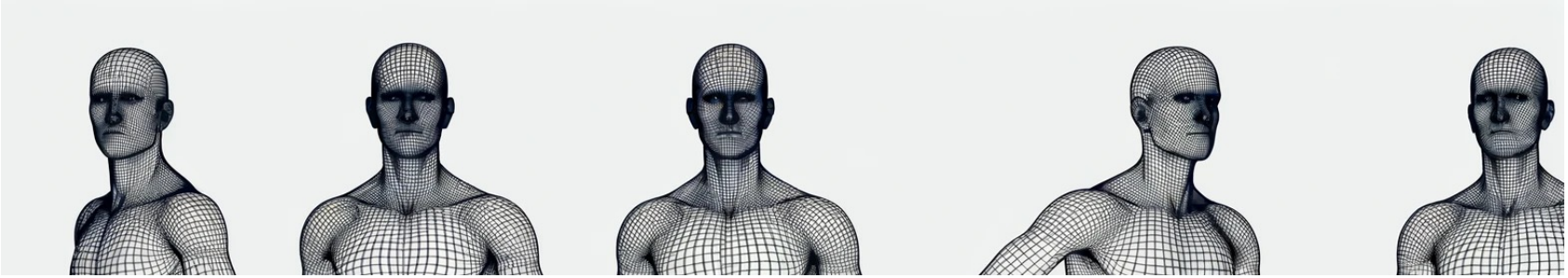
- So far, we talked about only static meshes... What about dynamic meshes?
- Animated meshes
- Sequence of mesh frames

Animated Mesh Compression

- **Sparse Keyframes**: Instead of storing every frame of the animation, only keyframes are stored, and intermediate frames are interpolated. This greatly reduces the amount of data required.
- **Interpolation**: The in-between frames are generated by interpolating the transformations (such as position, rotation, and scaling) from keyframes. Efficient algorithms ensure that this interpolation does not require too much computational power.

Compressing a mesh sequence

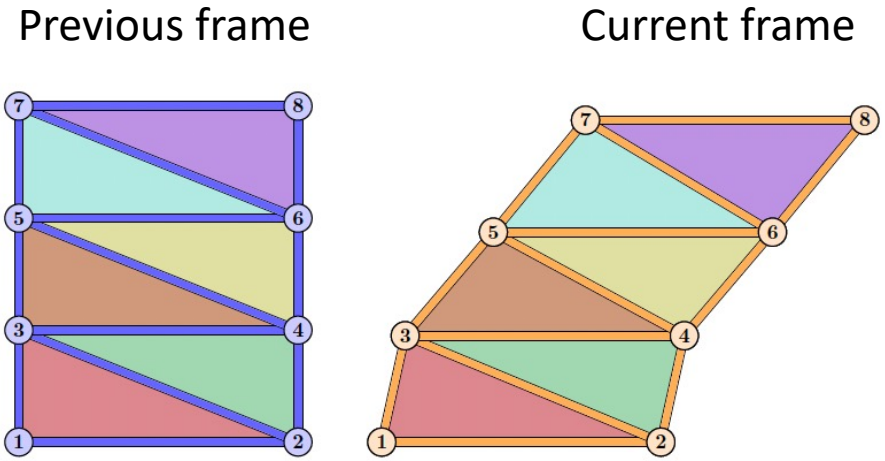
- Recall intra and inter frame prediction for exploiting spatial and temporal redundancy in 2D videos
  - Can we apply similar principles?



ChatGPT produced example mesh sequence

Compressing a mesh sequence

- Compress displacements instead of vertices
  - Displacements are much smaller values and require fewer bits compared to vertices

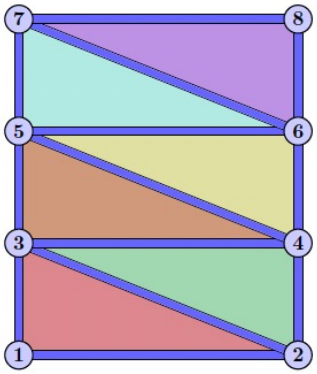


- Key assumption: vertex correspondence

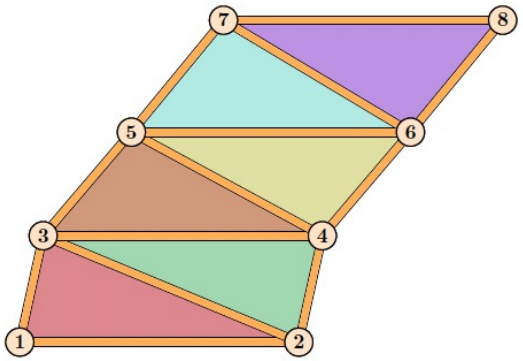
Compressing a mesh sequence

- Compress displacements instead of vertices
  - Displacements are much smaller values and require fewer bits compared to vertices

Previous frame



Current frame



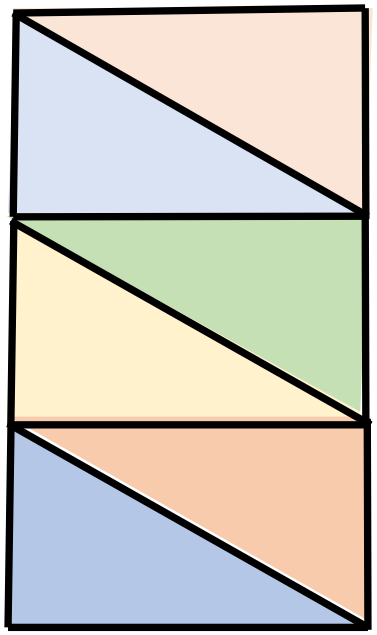
- Key assumption: vertex correspondence

Final Step: Entropy Coding

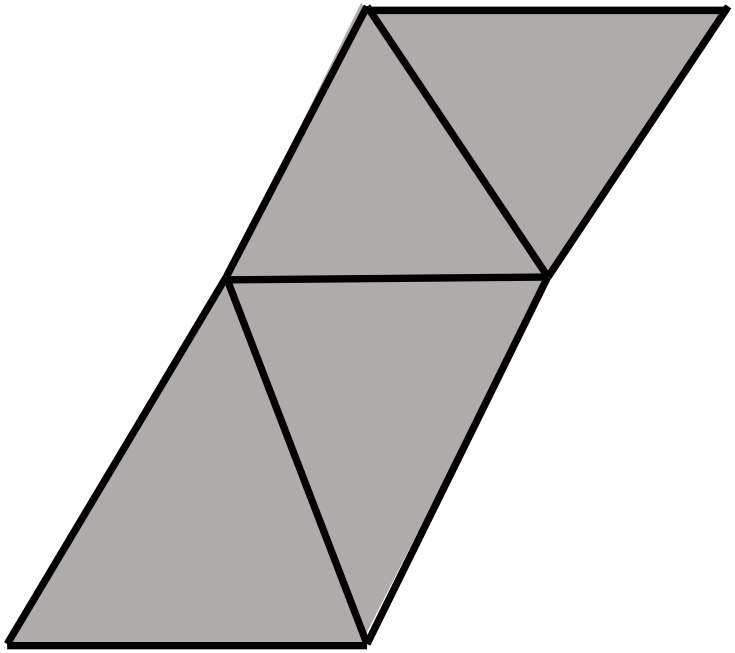
Compressing a mesh sequence

- Topology changes in practice (also called as time varying mesh)

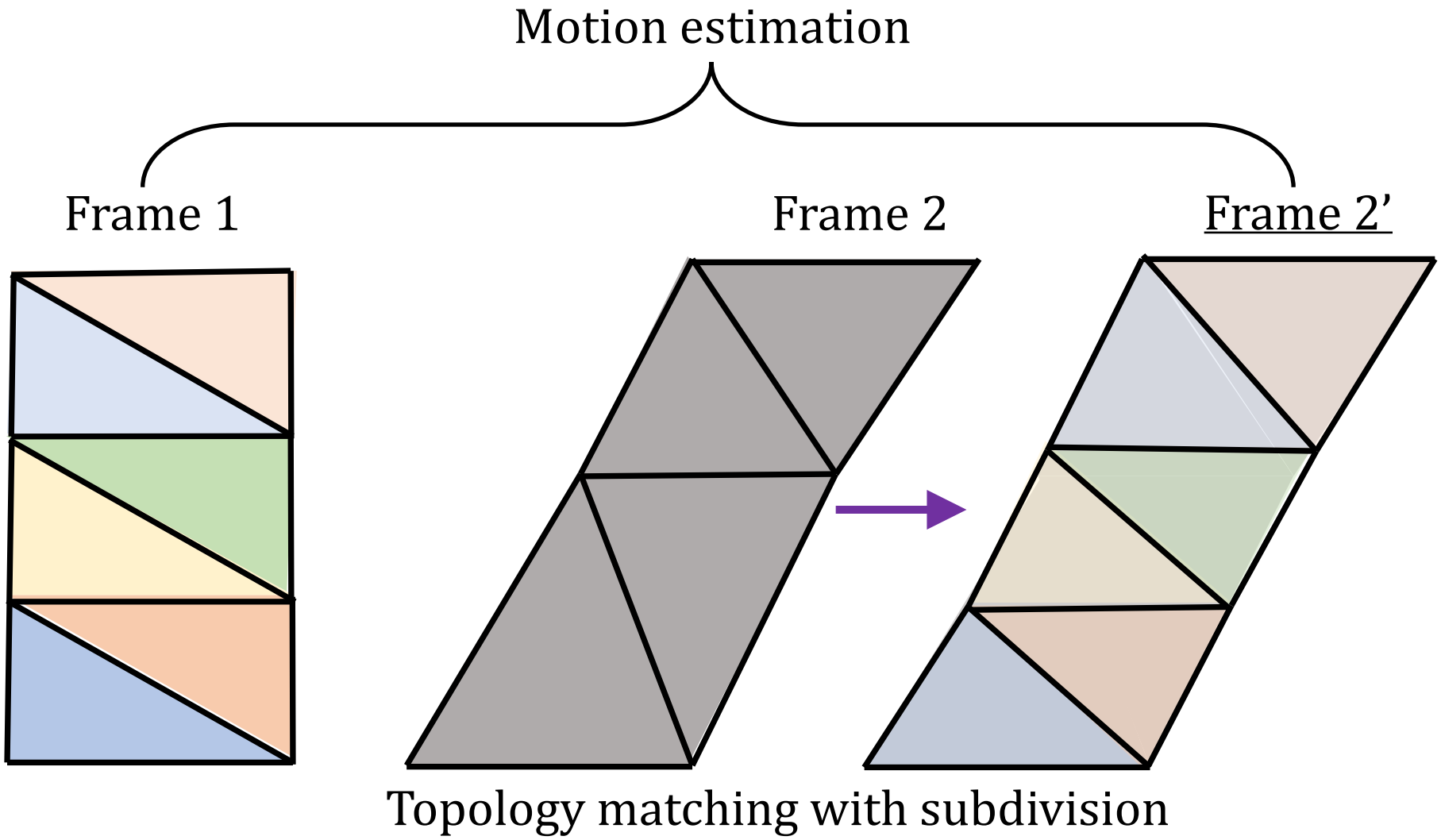
Frame 1



Frame 2



Compressing a mesh sequence



Compressing a mesh sequence

- Extract key points from each mesh
- Establish correspondences
- Apply non-rigid transformation

Open3D

Compressing a mesh sequence

- Challenges
  - Not easy to get a useful reference mesh always – due to self contact or addition or deletion of geometry across time

- Still an active area of research – no open source or very well adopted techniques yet

Compressing a mesh sequence

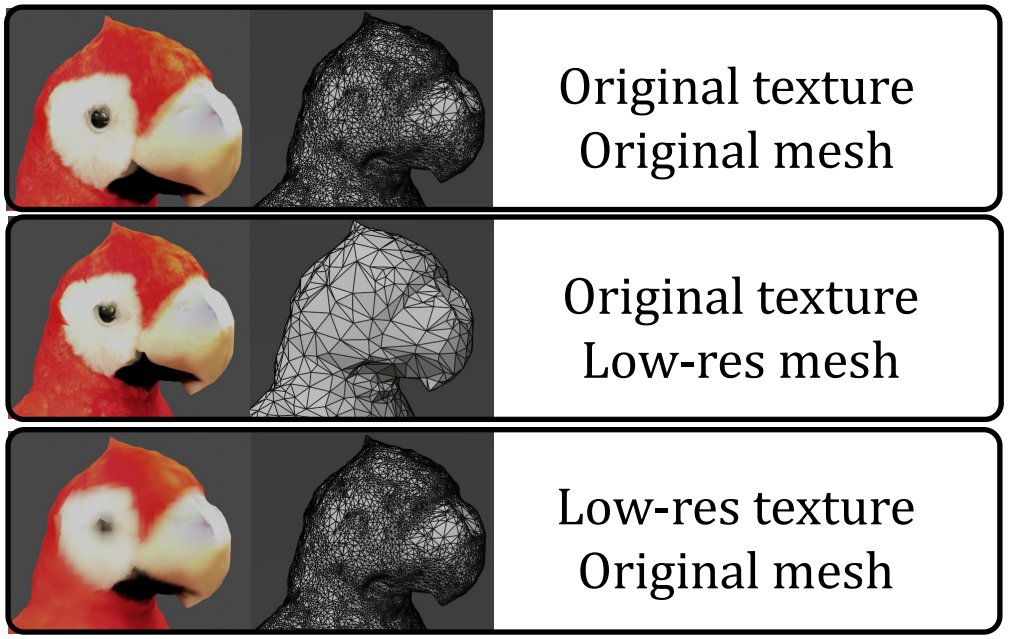
- Small-scale vs. Large scale mesh sequences
- Large meshes often tend to be static most of the regions
  - Divide the mesh into patches
  - Check if the patch is static, if so, don't store, just store a motion vector (note even in static cases topology can change but since we "know" it's a static region it's okay use the lossy reference)
  - If not static, need to store
  - **Question: how to detect if it's static or moving?**

Compressing mesh and texture together

- We care about the final rendered image quality – so we need to optimize a function that compresses both mesh texture and mesh together
  - Need to effectively allocate bits for mesh and texture

Compressing mesh and texture together

- We care about the final rendered image quality – so we need to optimize a function that compresses both mesh texture and mesh together
  - Need to effectively allocate bits for mesh and texture



Mesh Compression

- This lecture has focused mainly on compression efficiency
  - Almost all of the algorithms are computationally very expensive
  - None of the dynamic or time varying methods can run in real-time – so only suitable for offline stored applications

Summary of the Lecture

- Mesh compression
  - Vertex, connectivity, texture compression
  - Static
  - Dynamic or time varying
  - Progressive

Next up: multi-view compression