Introduction to Computer Graphics

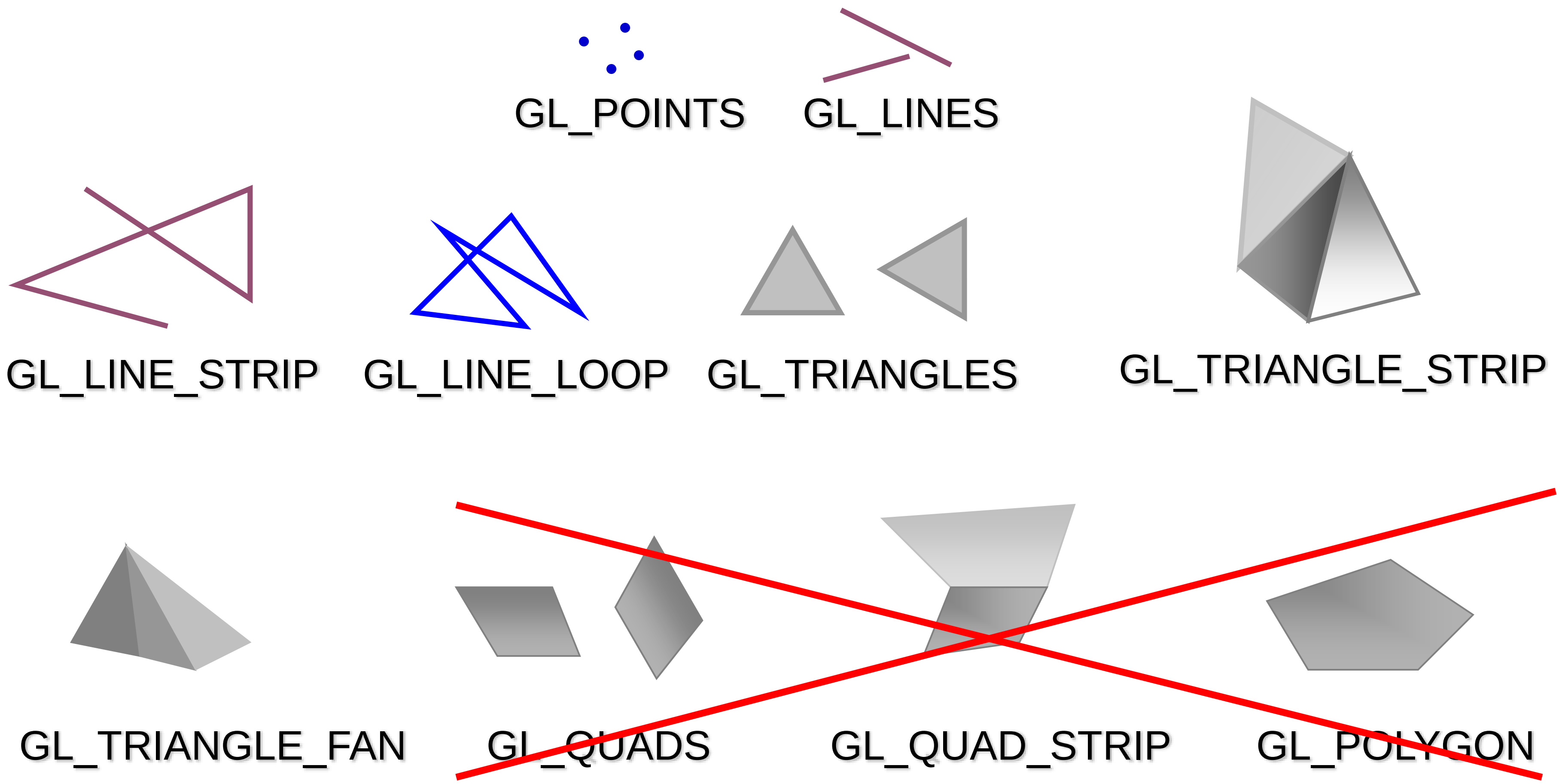
OpenGL Primitives

Prasanna Ghali

OpenGL Geometric Primitives (1/2)

- GL provides mechanisms to describe how complex geometric objects are to be rendered
- Doesn’t provide mechanisms to describe the complex objects themselves

OpenGL Geometric Primitives (2/2)



Why Triangle Primitives In Real-Time Graphics?

- Triangles provide cleanest possible definition of rendering primitive
  - Also used for rendering `GL_POINT` and `GL_LINE` primitives

Why Triangle Primitives In Real-Time?

- Can approximate any shape
  - Meshes generated by modeling software can be converted to triangular meshes
  - Scalable – can vary number and density of triangles
- Always convex
  - Line segment between two edges will always be interior
- Vertices are always coplanar
  - Simplest polygon defining plane equation: $Ax + By + Cz + D = 0$
- Linear interpolation is mathematically straightforward
  - Barycentric coordinates allow any attribute to be interpolated

Why Not Quads?

- Commonly used in subdivision surface modeling
- However, not suitable as rendering primitive
  - Can be concave
  - Can be self-intersecting
  - Four vertices need not be on same plane – not suitable for real-time lighting and texturing
  - Bilinear interpolation required for per-vertex attributes

Model Representations: Triangle Meshes

- Requires geometry data
  - Position coordinates
  - Topology - triangle connectivity information
- Requires attribute data
  - Normals
  - Texture coordinates
  - Surface material properties: degree of reflectivity, shininess, transparency, …
  - …

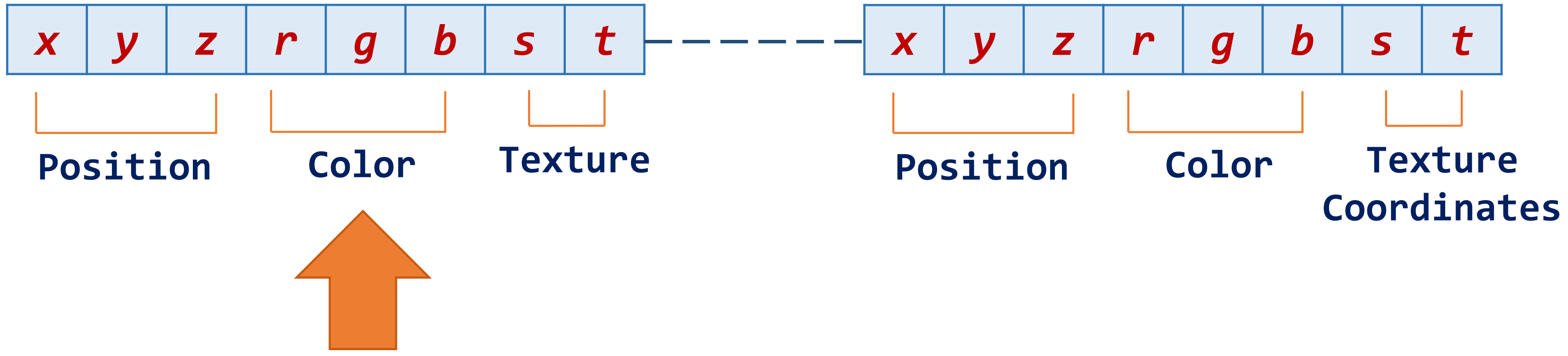
Triangle Mesh Representations

- Possible representations
  - Explicit (non-indexed)
  - Indexed
  - Explicit (non-indexed) fans and strips
  - Indexed fans and strips
- Tradeoffs in space and time
  - Storage size
  - Access time
  - Processing time

Triangle Mesh Layout in GPU Memory

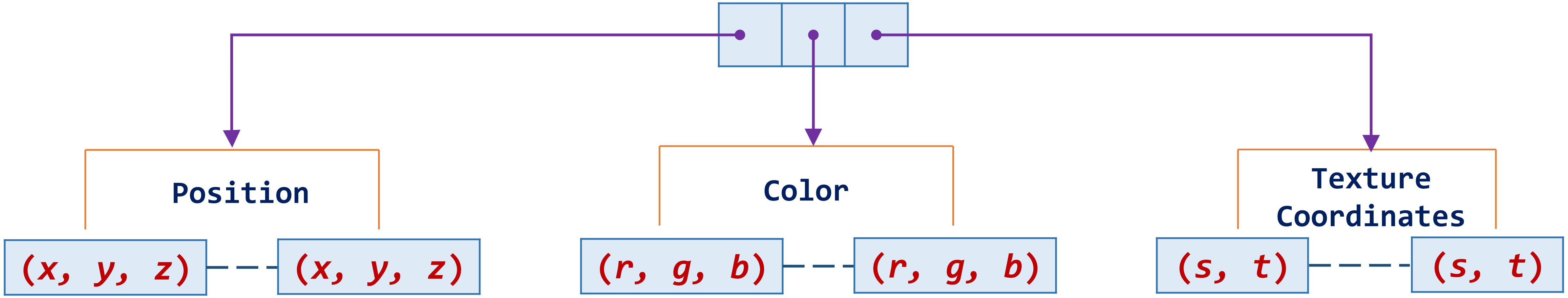
- Structure of arrays layout
- Array of structures layout

Array of Structures Layout



```
struct Vertex {
  glm::vec3 pos; // position
  glm::vec3 nml; // normal
  glm::vec2 tex; // texture coordinates
};
```

Structure of Arrays Layout



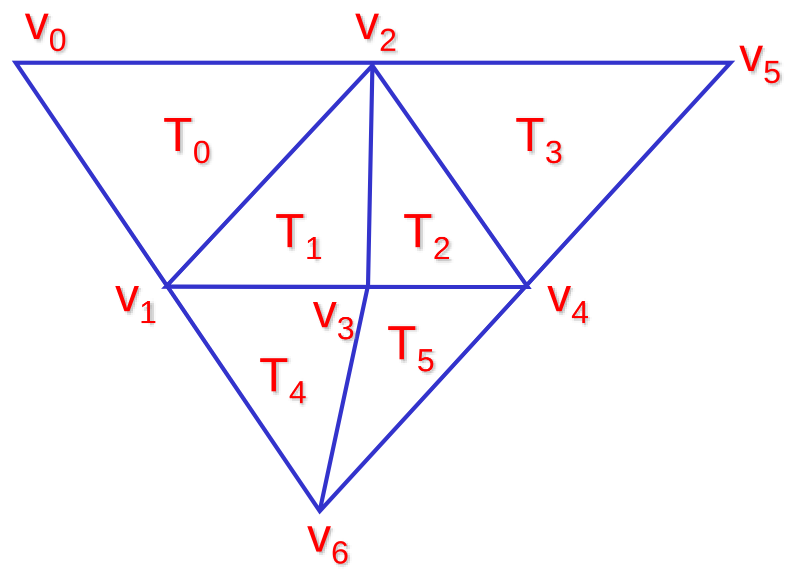
AoS vs SoA

- AoS more readable because of logical grouping
- AoS might require more memory because of padding compared to SoA
- Cache locality
  - AoS has better cache locality when dealing with entire vertex data
  - SoA has better cache locality when dealing with specific vertex attribute (and not entire vertex data)

Reference Mesh Model

- Reference model with 6 triangles and 7 vertices
- Each vertex modeled as

```
struct Vertex {
  glm::vec3 pos; // position
  glm::vec3 nml; // normal
  glm::vec2 tex; // texture coordinates
};
```



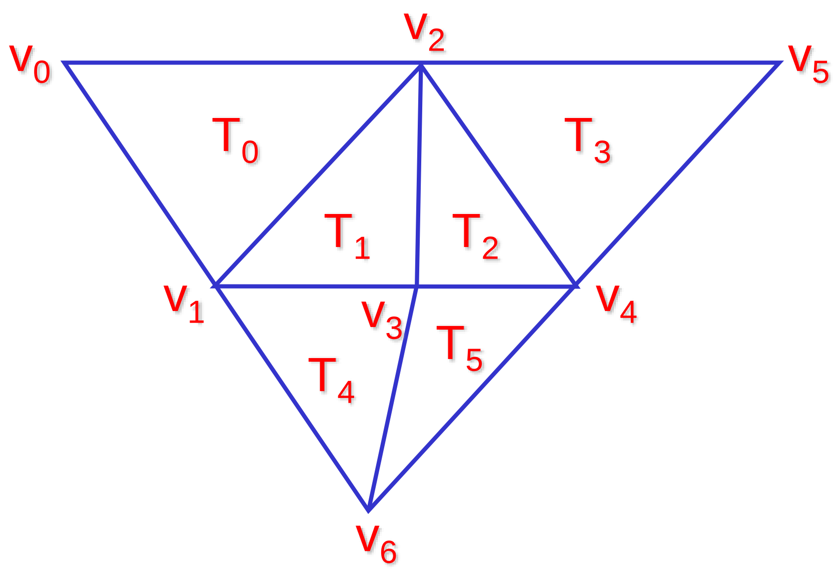
Non-Indexed (Explicit) Representation (1/4)

- Triangle represented as array of m elements with each element containing data for its 3 vertices

```
struct Triangle_Explicit {
  Vertex p0, p1, p2;
};

struct TriMesh_Explicit {
  GLint             tri_cnt;
  Triangle_Explicit *ptr;
};
```

Non-Indexed (Explicit) Representation (2/4)



v_0	v_1	v_2
v_1	v_3	v_2
v_3	v_4	v_2
v_4	v_5	v_2
v_1	v_6	v_3
v_3	v_6	v_4

Non-Indexed (Explicit) Representation (3/4)

- Assume model has m triangles and n vertices
  - By Euler's theorem, large triangular model typically has about twice more triangles than vertices, i.e., $m \approx 2n$
- Total memory requirement:
  - Explicit method requires array of m triangles
  - Each element in array contains data for 3 vertices
  - Each vertex requires 32 bytes
  - Total memory requirement: $3 \cdot 32 \cdot m \approx 96m$ bytes
- Total data transfer from buffer to vertex shader: $96m$ bytes

Non-Indexed (Explicit) Representation (4/4)

- Advantage
  - Simplicity?
- Disadvantages
  - High storage size – vertices stored multiple times
  - Vertex shader requires greater time to transform, light, and clip duplicated vertices
- Seldom used by application programmers for general meshes
  - But, many graphics drivers convert other representations into explicit stream when sending data to vertex shader

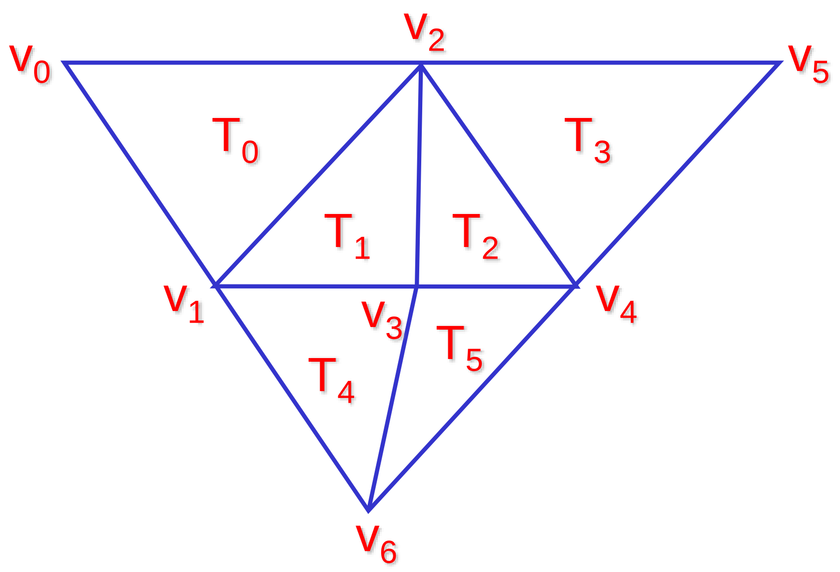
Indexed Representation (1/3)

- Each vertex stored once in a vertex list
- Triangles represented by list of indices into vertex array

```
struct Triangle_Indexed {
  // indices into vertex array
  GLushort i0, i1, i2;
};

struct TriMesh_Indexed {
  GLint vtx_cnt, tri_cnt;
  Vertex *p_vtxlist; // vertex list
  // triangle list
  Triangle_Indexed *p_trilist;
};
```

Indexed Representation (2/3)



v_0
v_1
v_2
v_3
v_4
v_5
v_6

T_0	0	1	2
T_1	1	3	2
T_2	3	4	2
T_3	4	5	2
T_4	1	6	3
T_5	3	6	4

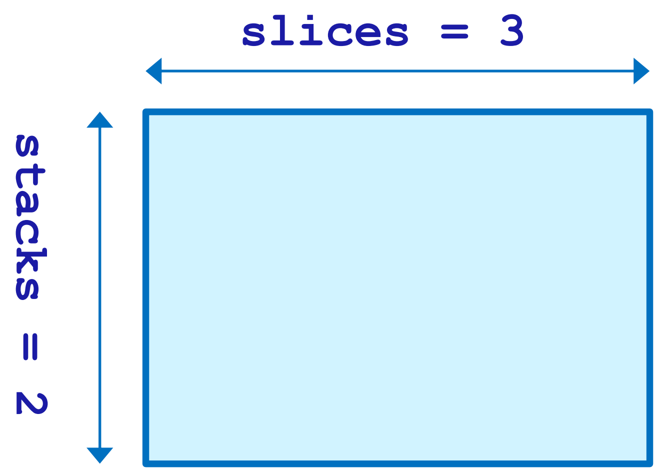
Indexed Representation (3/3)

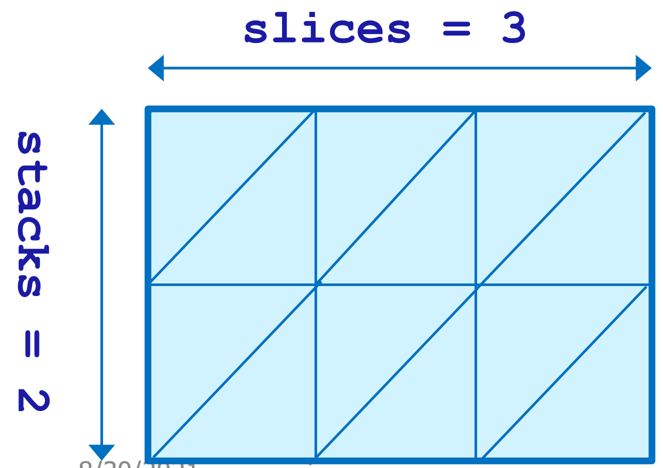
- Assume model has m triangles and n vertices
  - Use approximation $m \approx 2n$
- Total memory requirement:
  - Indexed method requires arrays of n vertices and m triangles
  - Each vertex requires 32 bytes
  - Each vertex index requires 2 bytes
  - Each element in triangle array refers to its 3 vertices using indices
  - Total memory requirement: $32{\cdot}n + 6{\cdot}m \approx 22m$ bytes
- However, data transfer is: $m{\cdot}3{\cdot}(2+32) = 102m$ bytes

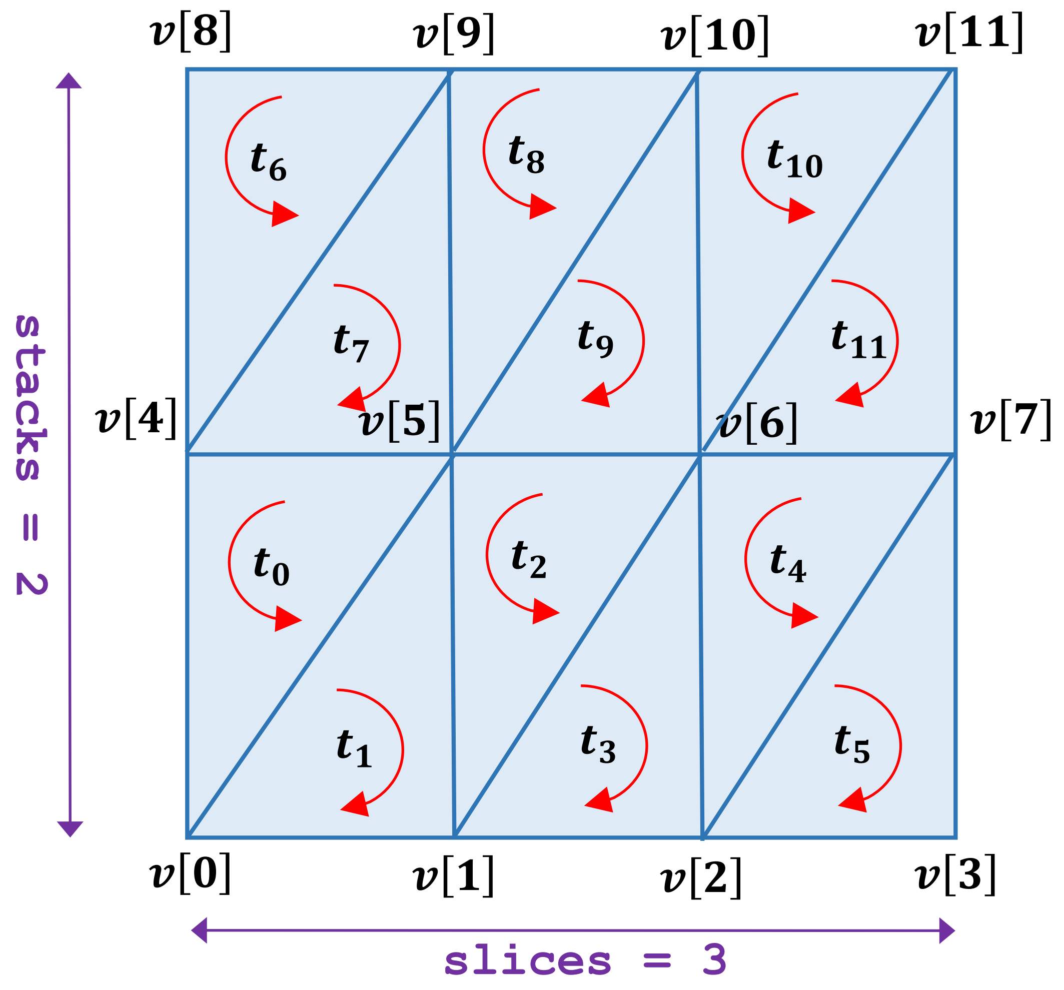
Triangle Strip Representation (1/5)

- Organization of triangular faces in mesh as sequence of contiguous triangles
- First triangle in strip requires three vertices
- Subsequent triangles use one additional vertex
- Requires GPU to have vertex cache for two vertices

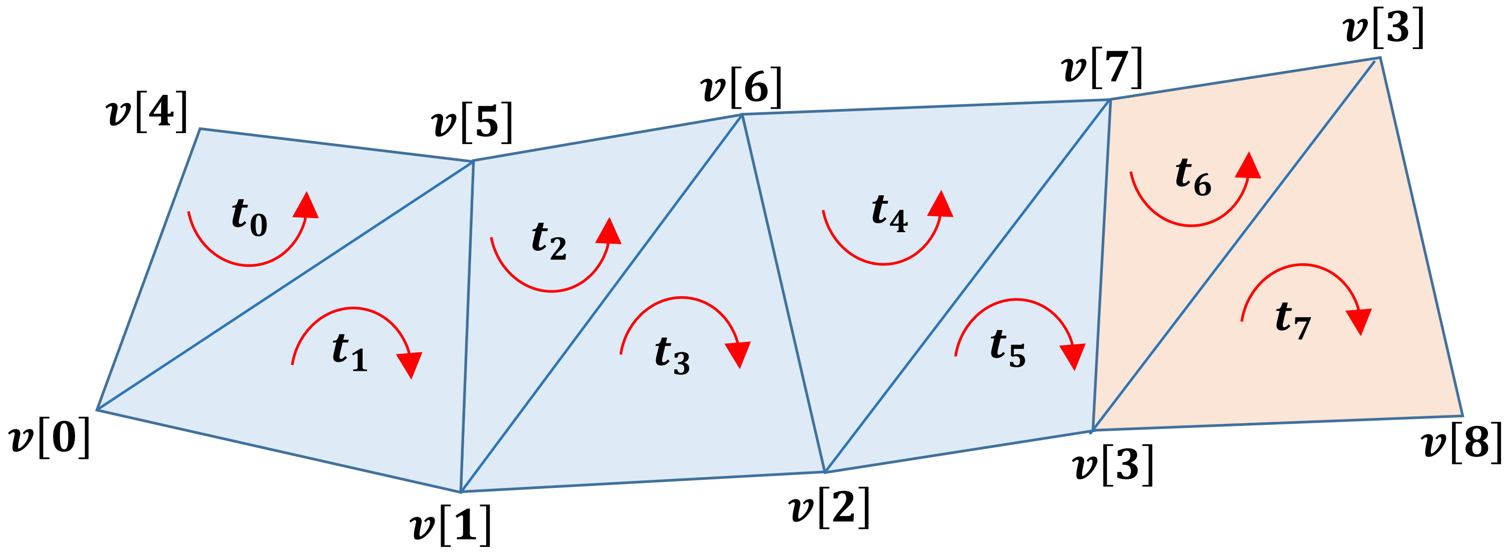
Triangle Strip Representation (2/5)

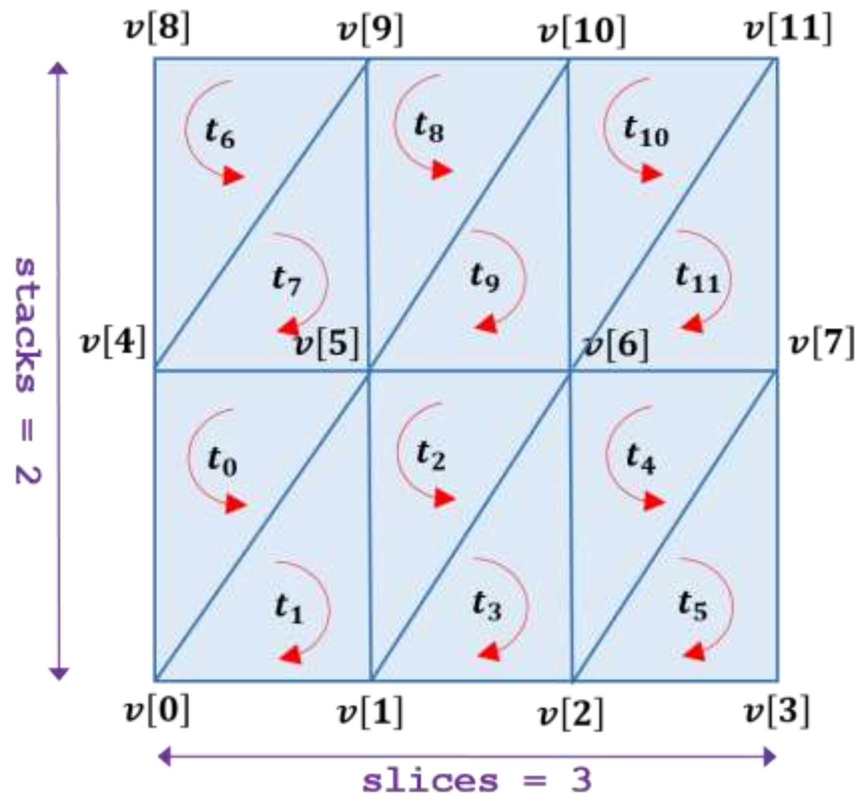






Triangle Strip Representation (3/5)

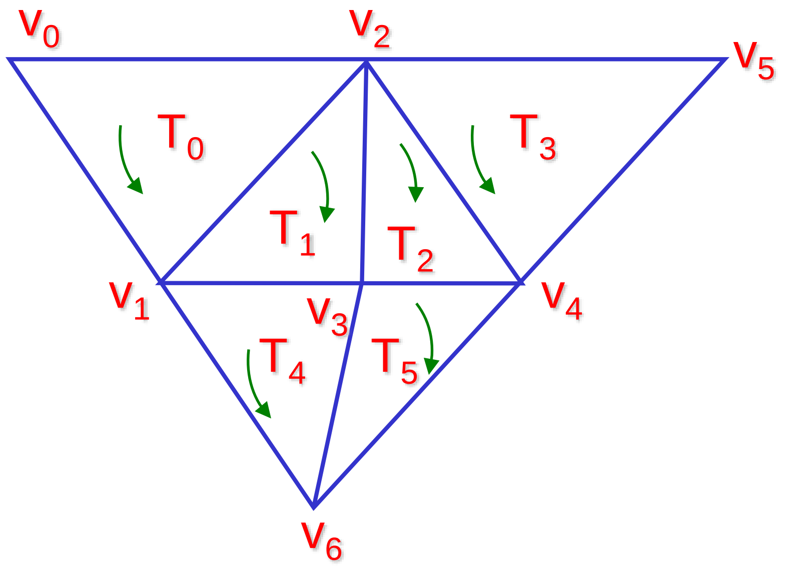




Triangle Strip Representation (4/5)

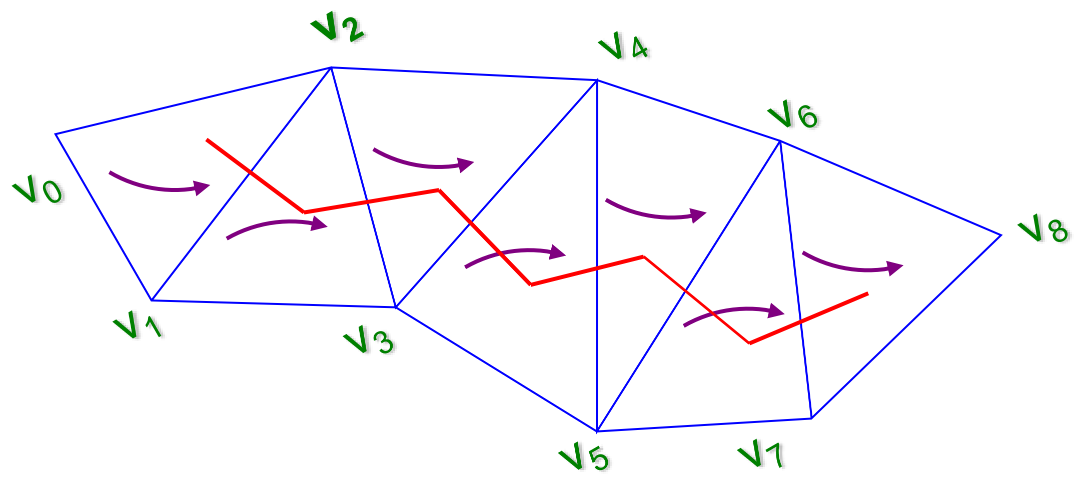
- Let b be strip bloat factor accounting for costs of restarting strips and overriding strip direction
  - Typically, $1.1 \leq b < 1.5$
- Overall size of representation is $32bm$
- Data transfer size: $32bm$ bytes

Triangle Strip Representation (5/5)



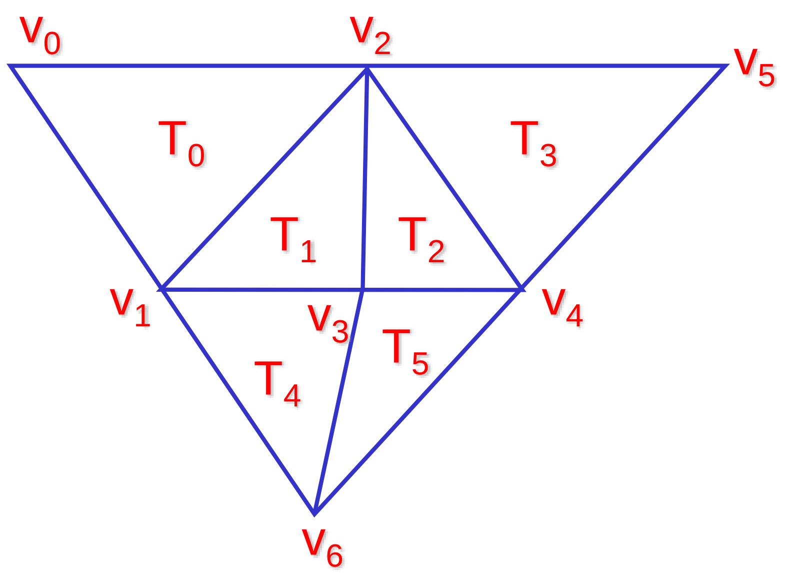
v_0	v_1	v_2 T_0
v_3 T_1	v_2	v_4 T_2
v_5 T_3	-1	

v_1	v_6	v_3 T_4
v_4 T_5	-1	



Indexed Triangle Strip Representation (1/2)

- Again mesh consists of vertex array and triangles that refer to these vertices through indices
- Except, triangles are organized into strips



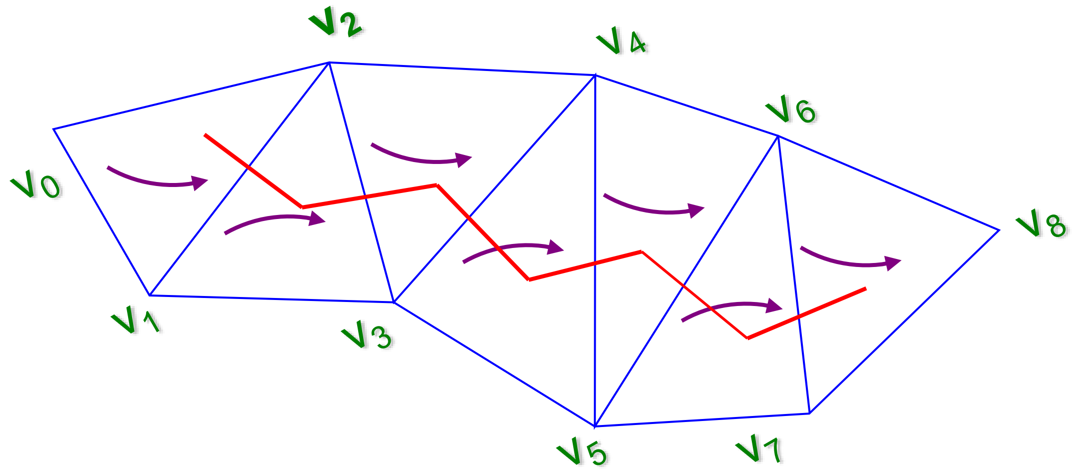
v_0
v_1
v_2
v_3
v_4
v_5
v_6

0	1	2 T_0
3 T_1	2	4 T_4
5 T_3	–1	

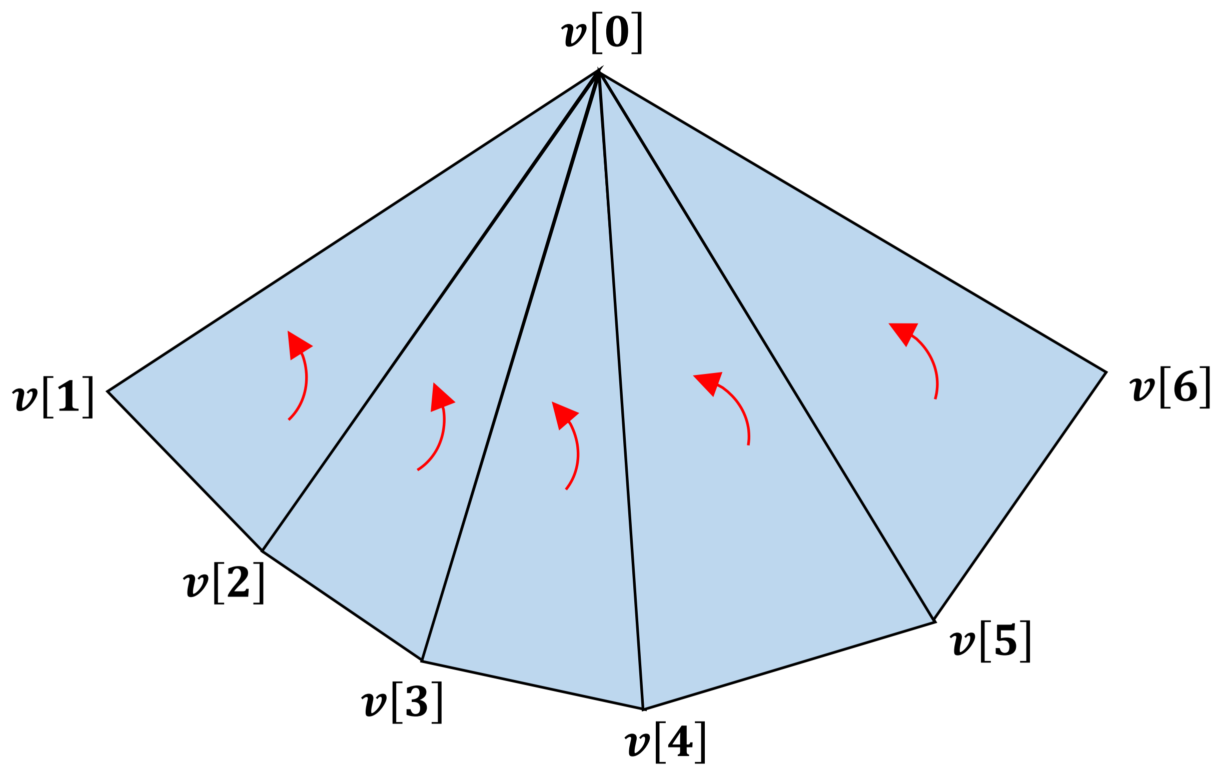
1	6	3 T_4
4 T_5	–1	

Indexed Triangle Strip Representation (2/2)

- Memory use: $32 \cdot n + 2 \cdot b \cdot m \cong (16 + 2b)m$ bytes
- Data transfer size: $34bm$ bytes



Triangle Fan Representation



Which Representation is Better?

Mesh Organization	Memory Size	Transfer Size
Explicit Triangles	$96m$	$96m$
Indexed Triangles	$22m$	$102m$
Triangle Strips	$\cong 32bm$	$\cong 32bm$
Indexed Triangle Strips	$\cong (16 + 2b)m$	$\cong 34bm$