M30242-Graphics and Computer Vision

Lecture 04: Triangle Strips and Degenerated Triangles

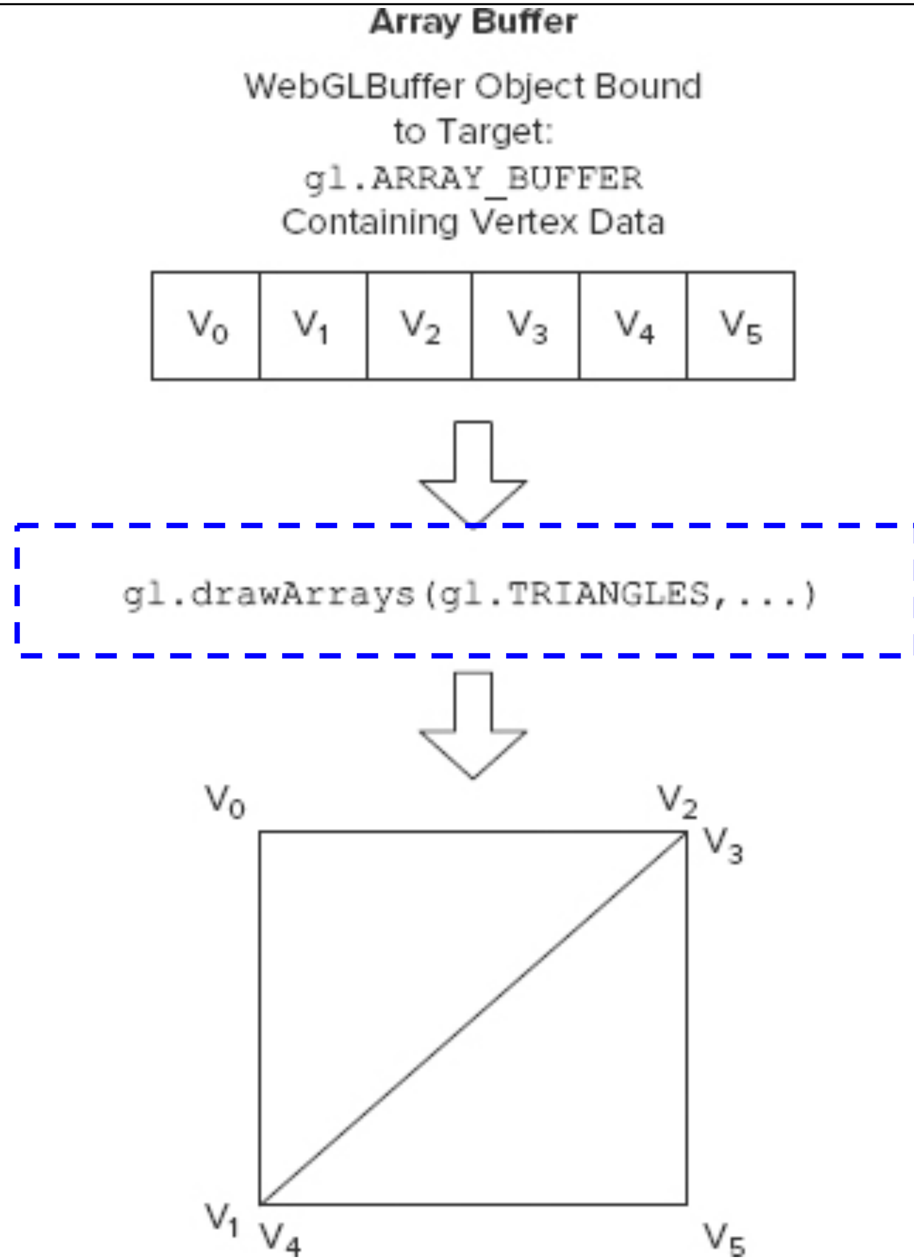
Overview

- Draw triangle strips (`gl.TRIANGLE_STRIP`)
  - The order of the vertex coordinates or indices in the buffers
- Degenerated triangles
- An example – draw a sphere

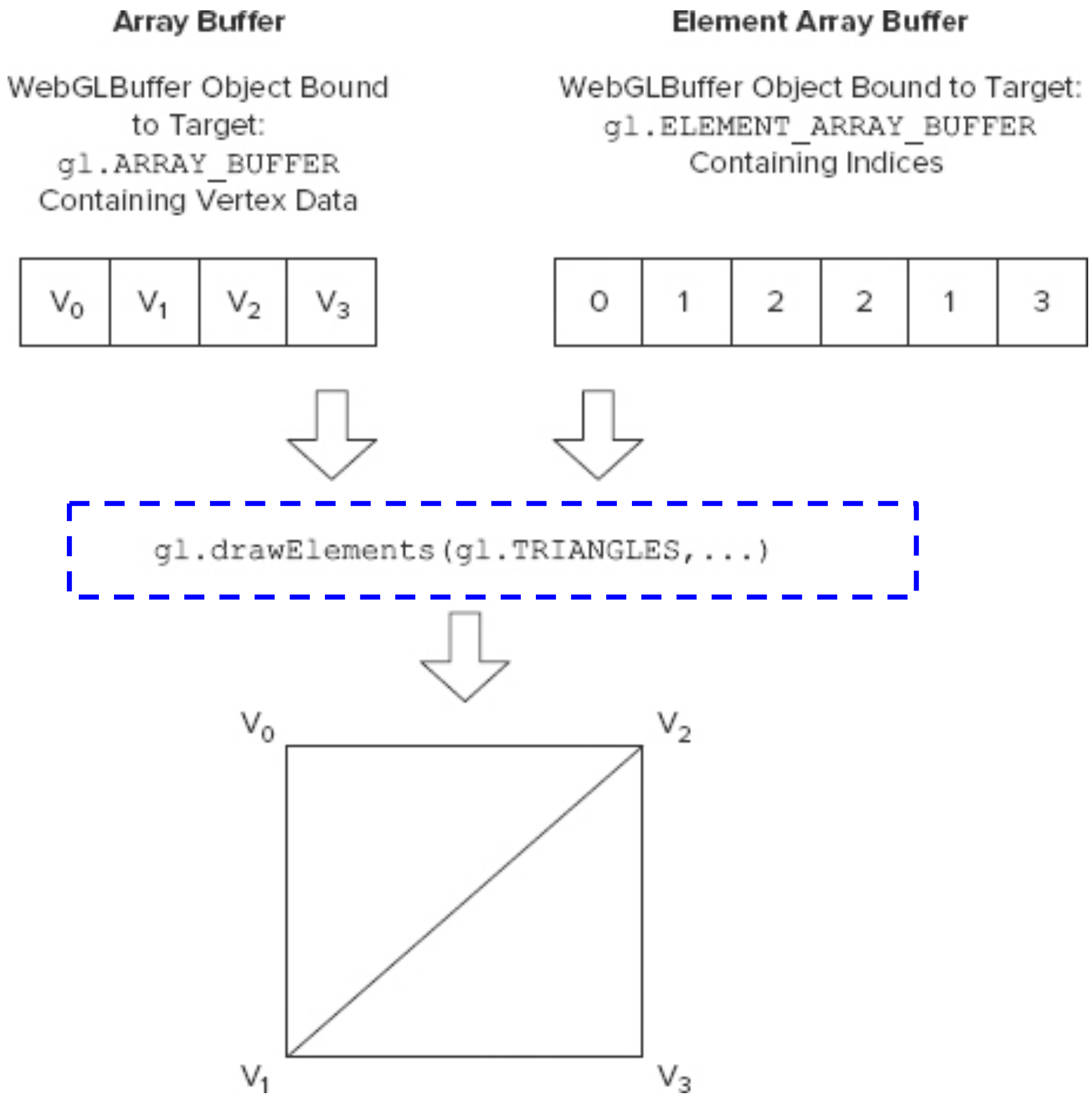
A Quick Recap

- In last lecture, we have learnt two ways to draw:
  - using **drawArrays()** – drawing from the vertex coordinates directly
  - using **drawElements()** – drawing from vertex indices
- In both cases, we considered drawing individual triangles (as **`gl.TRIANGLES`**), therefore the vertex coordinates or indices that define a triangle are given explicitly – there is no restriction on the order of the coordinates or indices being stored in the buffer.

```
void drawArrays(GLenum mode, GLint first, GLsizei count)
```



```
void drawElements(GLenum mode, GLsizei count, GLenum
  type, GLintptr offset)
```



Draw Triangle Strips

- We can draw triangle strips by setting the drawing mode flag **`gl.TRIANGLE_STRIP`** in

  drawArrays() or

  drawElements()

  functions.
- When in **`gl.TRIANGLE_STRIP`** mode, the order of the vertex coordinates and indices in the buffers is important.
- Instead of specifying triangles by the programmer, WebGL constructs triangles automatically by fetching vertex data from the buffers in a *predefined* order.

Draw Triangle Strips

- Consider the following vertex data in data buffer (vertex coordinate buffer if `drawArrays()` is used, or index buffer if `drawElements()` is used),

1	2	3	4	5	6	7	8	…
A	B	C	D	E	F	G	H	…

- WebGL pipeline will use them in the following order to construct triangles:

  ABC: first triangle

  CBD: Drop the first item (A), swap the remaining 2 and take a new item (D)

  CDE: Drop the 2nd item (B) and take a new item (E)

  EDF: Drop the 3rd item (C), swap the remaining 2 and take a new item (F)

  EFG: Drop the 4th item (D) and take a new item (G)

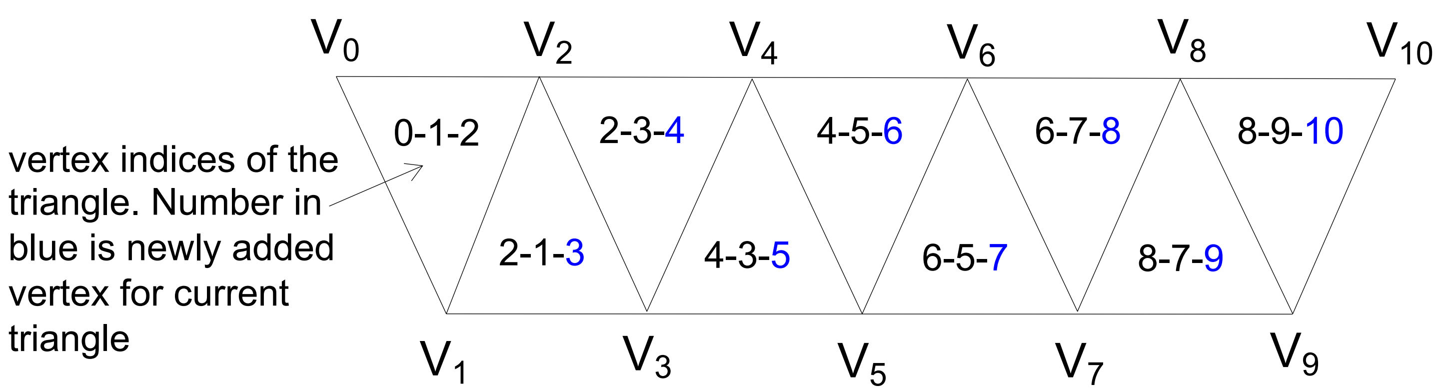
  . . .

Vertex Order

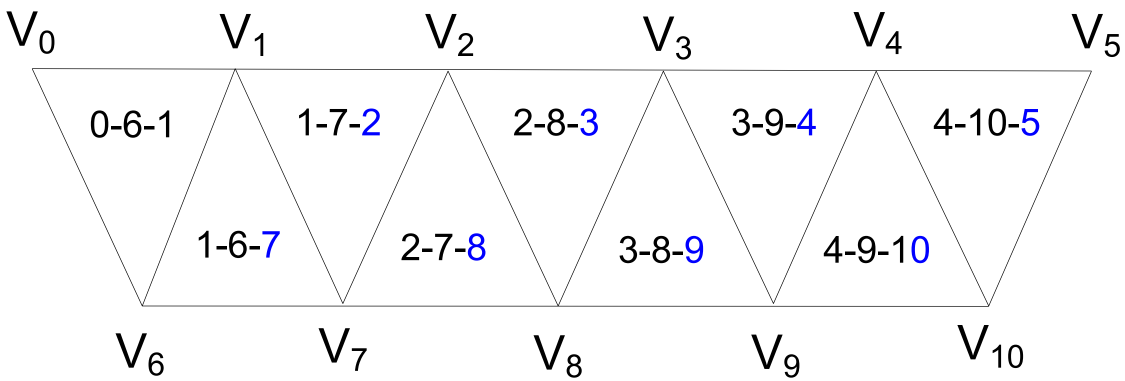
- For example, in the following strip, if we index the vertices as shown (and store them in the vertex coordinate data buffer in that order) and put 0,1,2,3,4,…10 into the index buffer, then all the triangles will be constructed automatically by calling

  `drawElements (gl.TRIANGLE_STRIP, …),`

  and all triangles have CCW winding.

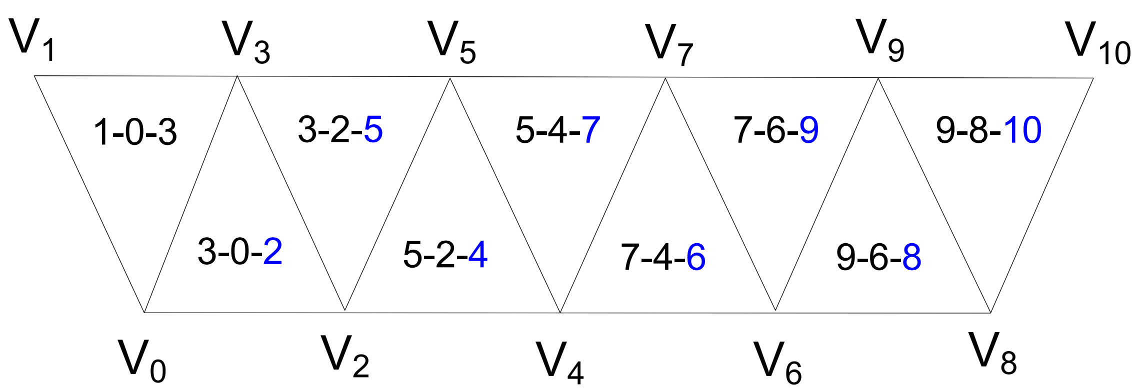


- If we index and store the vertices coordinate in a different order, to draw the strip correctly (using `drawElements()`) we need to arrange the indices in the index array buffer accordingly:



The order of index should be: **0,6,1,7,2,8,3,9,4,10,5**

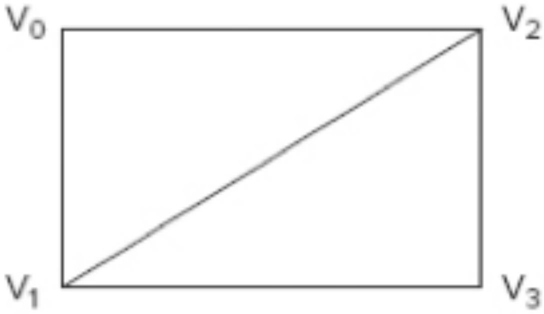
or



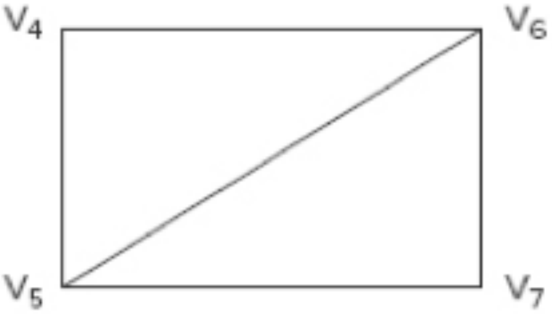
The order of index should be: **1,0,3,2,5,4,7,6,9,8,10**

Degenerated Triangles

- From the performance point of view, we wish to make as few calls as possible to functions `gl.drawArrays()` or `gl.drawElements()`.
- Suppose we have several independent triangle strips to draw, how can we combine the strips and draw them in one call to the drawing functions?



Independent strip 1



Independent strip 2

Degenerated Triangles

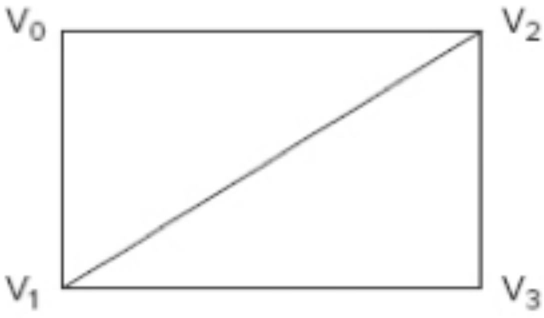
- The solution to this discontinuity or jump between strips is to insert extra (duplicated) vertices (`gl.drawArrays()` is used) or indices (`gl.drawElements()`).
- The inserted extra vertices/indices result in degenerated triangles.
- A degenerated triangle has *at least two indices* (or vertices) that are the same, and therefore the triangle has *zero area*.
- Degenerated triangles are easily detected by the GPU and discarded.

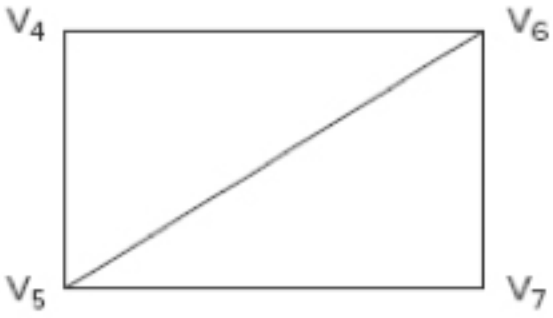
Repeat Vertices

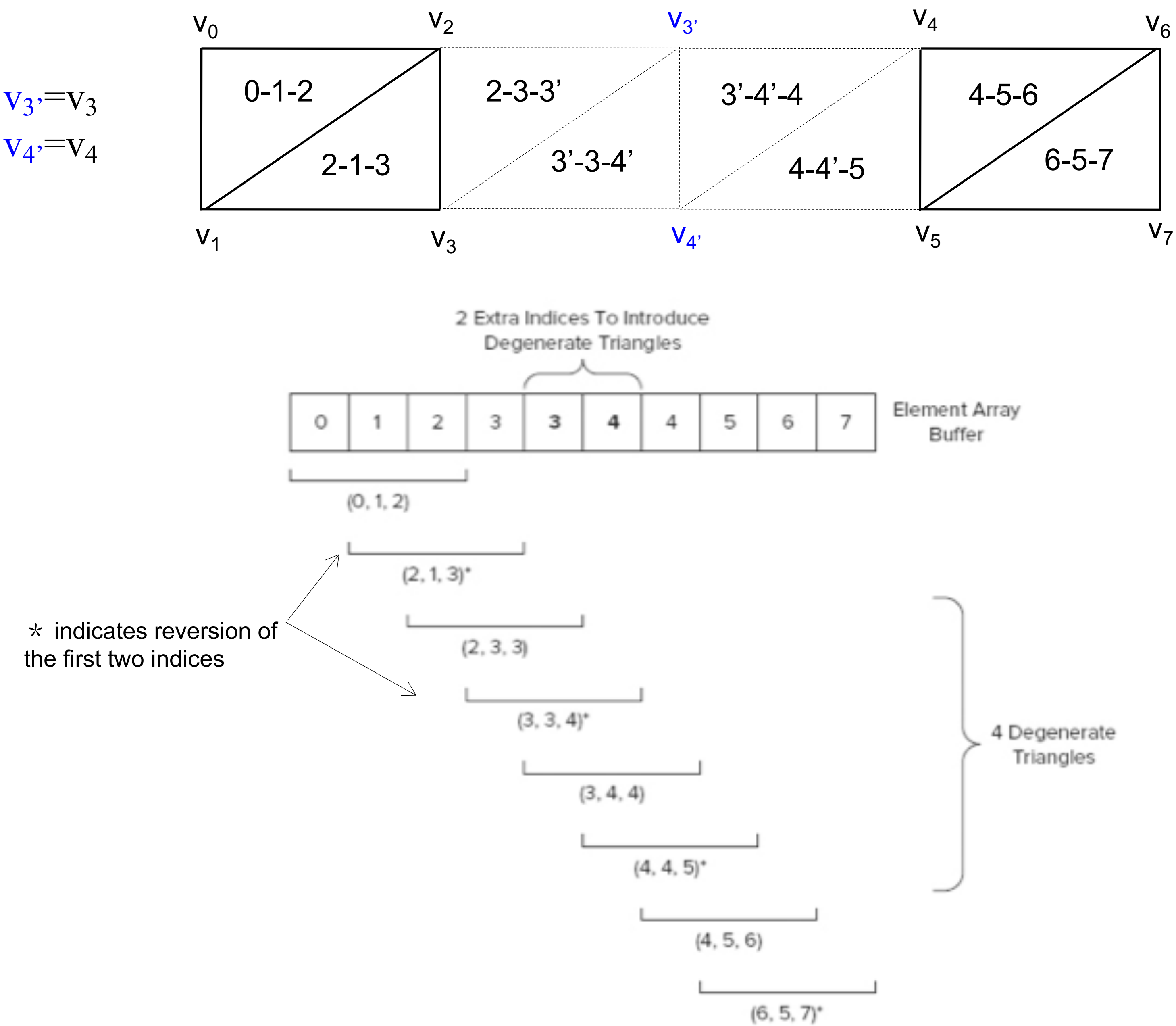
- The number of extra vertices (indices) needed for connecting two triangle strips depends on the number of triangles in the first strip.
- This is to ensure all the triangles of the two strips to have a consistent (same) winding order of the vertices. (still remember the importance of the winding order?)
- We have two cases:
  - Case 1: The first strip consists of an even number of triangles. Two extra indices are needed.
  - Case 2: The first strip consists of an odd number of triangles. Three extra indices are needed
- (For the convenience of discussion, in our exampes we will use indices to illustrate the degenerated triangles)

Case 1

- The first strip consists of an even number of triangles, e.g., it has 2 triangles:
  - The two triangles in 1st strip, (V0, V1, V2) and (V2, V1, V3), correspond to the indices (0, 1, 2, 3) in the element array buffer.
  - The second strip also consists of 4 vertices. The two triangles in 2nd strip, (V4, V5, V6) and (V6, V5, V7), correspond to the indices (4, 5, 6, 7) in the element array buffer.
- To connect the two strips and to keep the same winding order for both strips, the last vertex of the first strip and the first of the second strip are repeated in the index buffer.







$v_{3'}=v_3$
$v_{4'}=v_4$
v_0
v_2
$v_{3'}$
v_4
v_6
0-1-2
2-1-3
2-3-3'
3'-3-4'
3'-4'-4
4-4'-5
4-5-6
6-5-7
v_1
v_3
$v_{4'}$
v_5
v_7
2 Extra Indices To Introduce Degenerate Triangles
0 1 2 3 3 4 4 5 6 7
Element Array Buffer
(0, 1, 2)
(2, 1, 3)*
(2, 3, 3)
(3, 3, 4)*
(3, 4, 4)
(4, 4, 5)*
(4, 5, 6)
(6, 5, 7)*
* indicates reversion of the first two indices
4 Degenerate Triangles

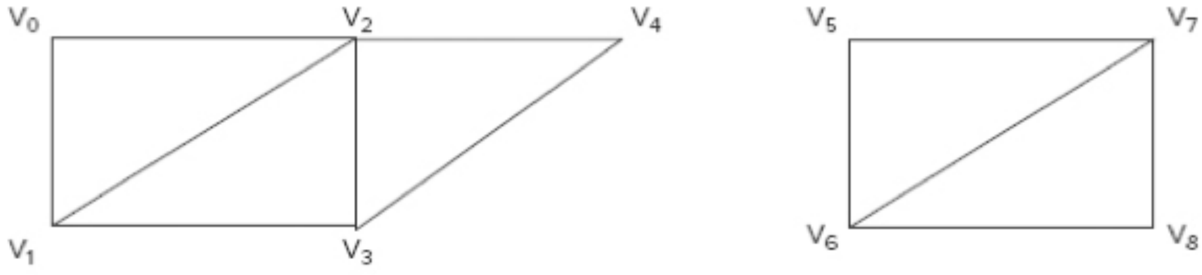
Cont’d

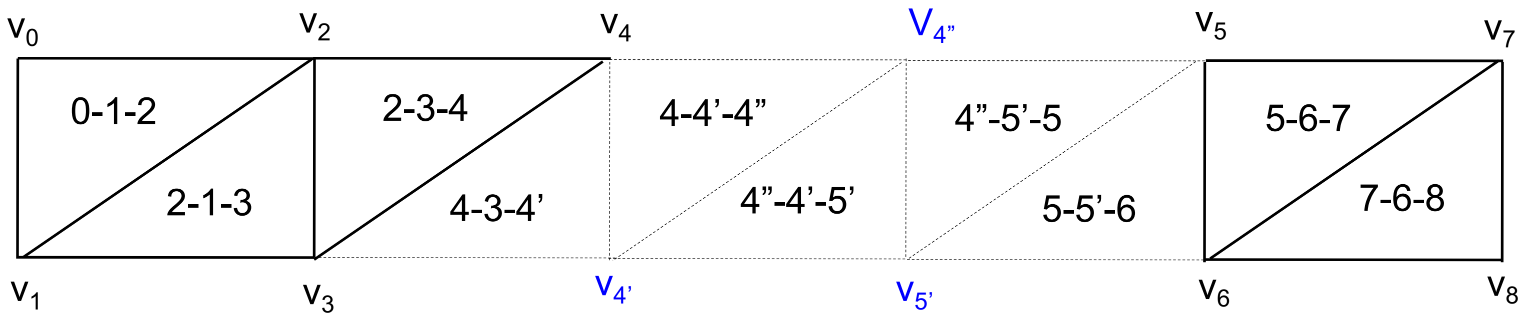
- The indices in the element array buffer will be interpreted as a single strip:
  - First you have the two triangles of the first triangle strip with index (0, 1, 2) and (2, 1, 3),
  - then you have the four degenerate triangles, (2,3,3), (3,3,4), (3,4,4) and (4,4,5) that will be discarded by the GPU, and
  - finally you have the two triangles of the last strip with indices (4, 5, 6) and (6, 5, 7).

- Question: what if we swap the positions of vertices $V_{3'}$ and $V_{4'}$?

Case 2

- The first strip consists of an odd number of triangles,
  - The first strip consists of the three triangles (V0, V1, V2), (V2, V1, V3), and (V2, V3, V4). They correspond to the element indices (0, 1, 2, 3, 4) in the element array buffer.
  - The second strip consists of triangles (V5, V6, V7) and (V7, V6, V8), corresponds to the element indices (5, 6, 7, 8) in the element array buffer.
- In this case, we need to add 3 extra indices to link them as a single strip: repeat the last vertex of the first strip twice and the first vertex in the second strip once.

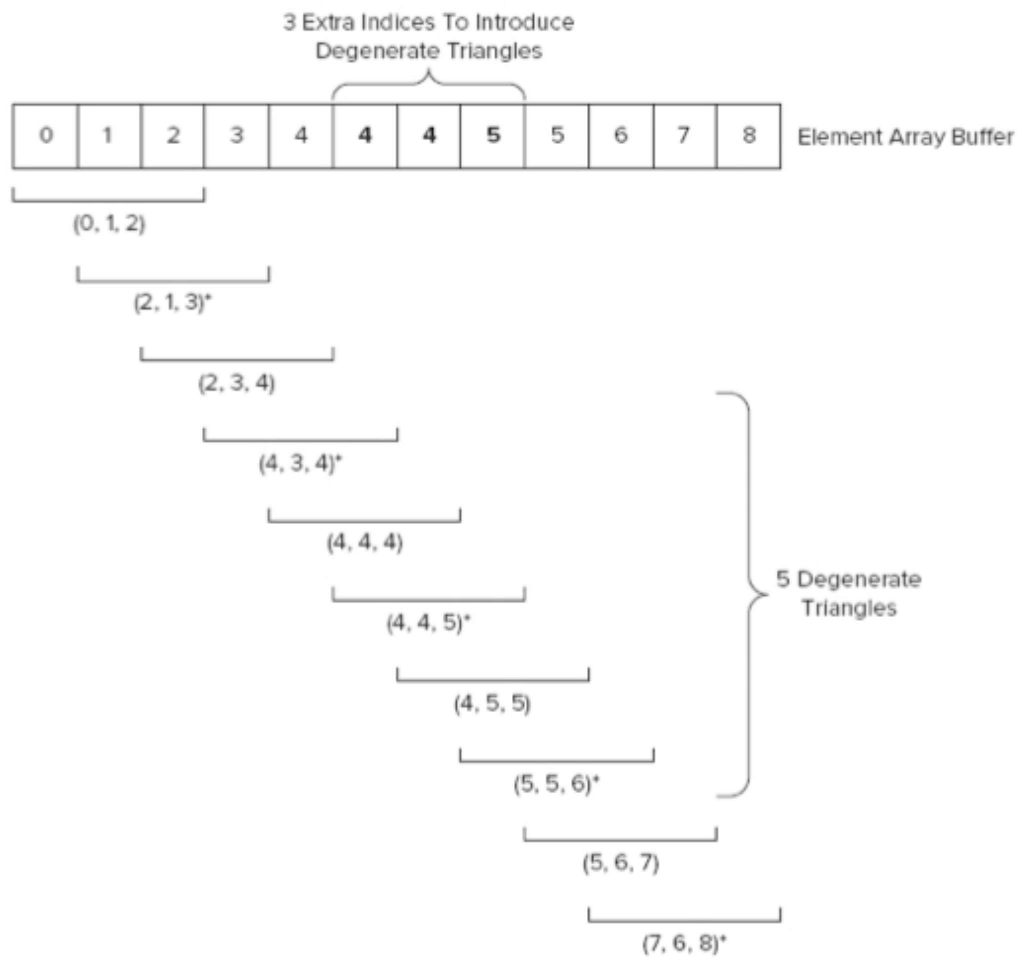




v0
v2
v4
V4”
v5
v7
0-1-2
2-1-3
2-3-4
4-3-4’
4-4’-4”
4”-4’-5’
4”-5’-5
5-5’-6
5-6-7
7-6-8
v1
v3
v4’
v5’
v6
v8

$v_{4'} = v_{4''} = v_4$

$v_{5'} = v_5$



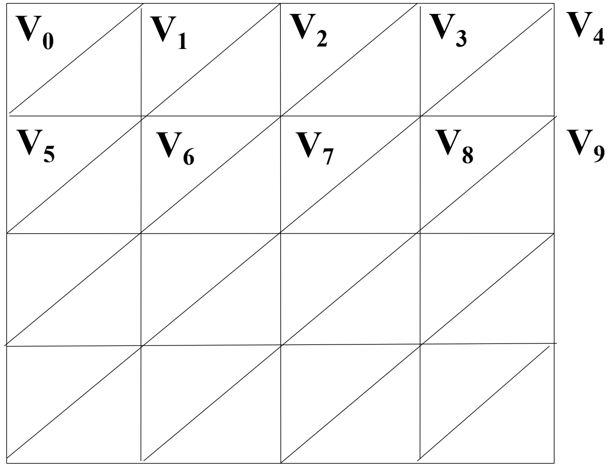
3 Extra Indices To Introduce Degenerate Triangles
0
1
2
3
4
4
4
5
5
6
7
8
Element Array Buffer
(0, 1, 2)
(2, 1, 3)*
(2, 3, 4)
(4, 3, 4)*
(4, 4, 4)
(4, 4, 5)*
(4, 5, 5)
(5, 5, 6)*
(5, 6, 7)
(7, 6, 8)*
5 Degenerate Triangles

Cont'd

- The indices will be interpreted as triangles in a single triangle strip.
  - First you have the three triangles of the first triangle strip with index (0, 1, 2), (2, 1, 3), and (2, 3, 4).
  - Then you have the five degenerated triangles that will be removed by the GPU.
  - Finally, you have the two triangles of the last strip with index (5, 6, 7) and (7, 6, 8).

Use of Triangle Strips

- Triangle strip is most efficient/convenient for drawing long ribbon-like shapes.
- But it can also be used to draw a mesh like the following:



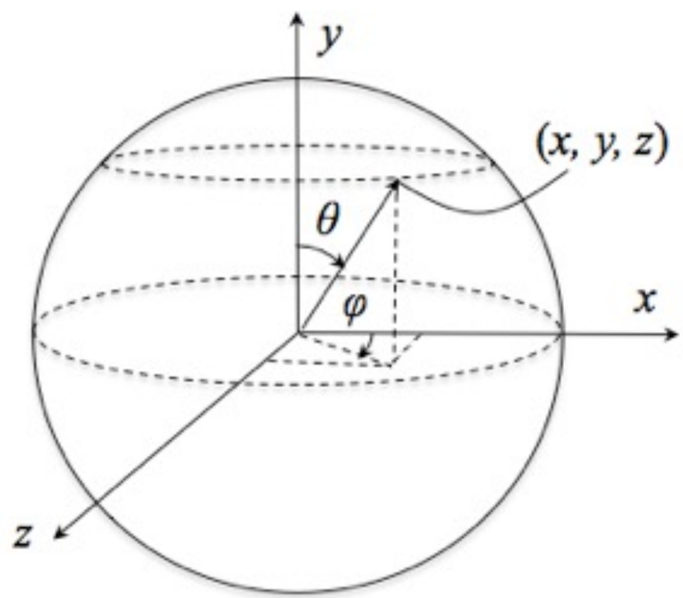
- Here, we can treat the mesh as 4 horizontal (or vertical) strips, and draw the entire mesh in one call. We have to use degenerated triangles at the end of the first 3 strips.

Tessellation – Acquiring Triangle Mesh

- Methods for obtaining mesh data:
  - Manual assignment – simple shapes (e.g., shapes used in the tutorials)
  - Calculation – for parametric surfaces. Graphics library usually provide calls to draw primitives such as sphere, cylinder, cone, etc.
  - Data exported from 3D graphics software (3DS Max, Maya).
  - Scanning – for complex freeform objects.

Triangulation of Spheres

- To draw a sphere, the sphere needs to be triangulated into triangle mesh consisting of vertex and normal data.
- A suitable coordinate system has to be used to facilitate the calculation.
  - For spheres – spherical coord. Three parameters: r [0, R], θ [0, π] and φ [0, 2π]
  - For cylinders – cylindrical coord. Three parameters: r [0, R], z [0, H] and φ [0, 2π]



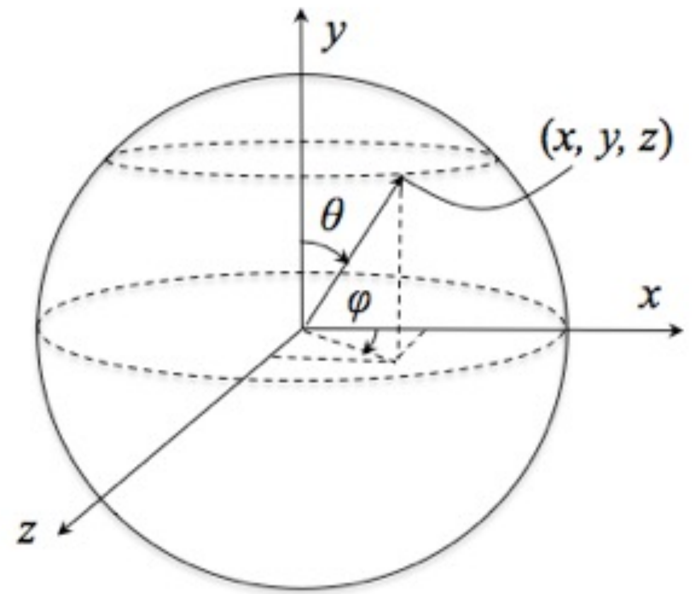
Sphere in Spherical Coord.

- In spherical coordinate system, a sphere take the form:

  $$x = r \sin\theta \cos\varphi$$
  $$y = r \cos\theta$$
  $$z = r \sin\theta \sin\varphi$$

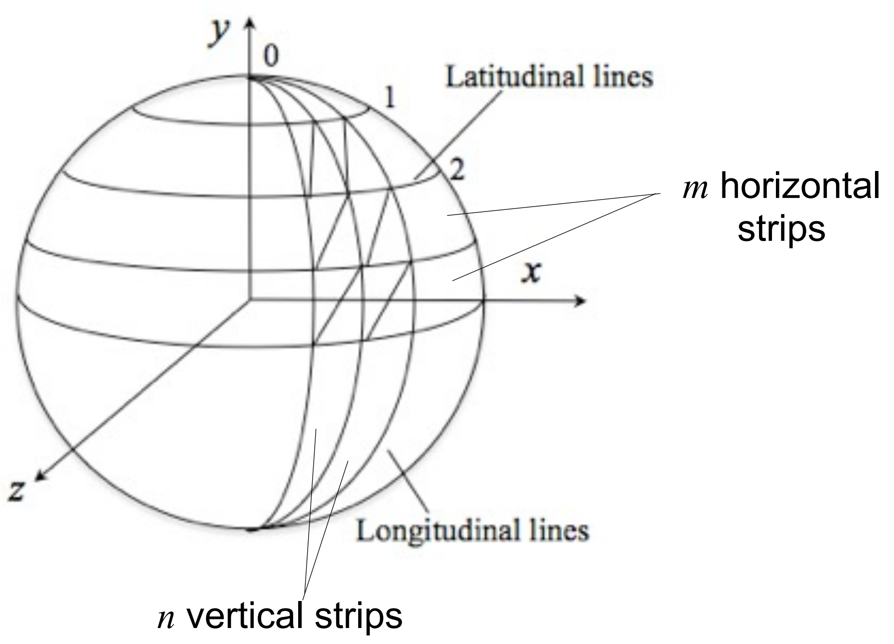
  where r is the radius of the sphere, and with θ: [0, π] and φ: [0, 2π]

- The coordinates of each and every points on the sphere can be calculated by this set of functions.



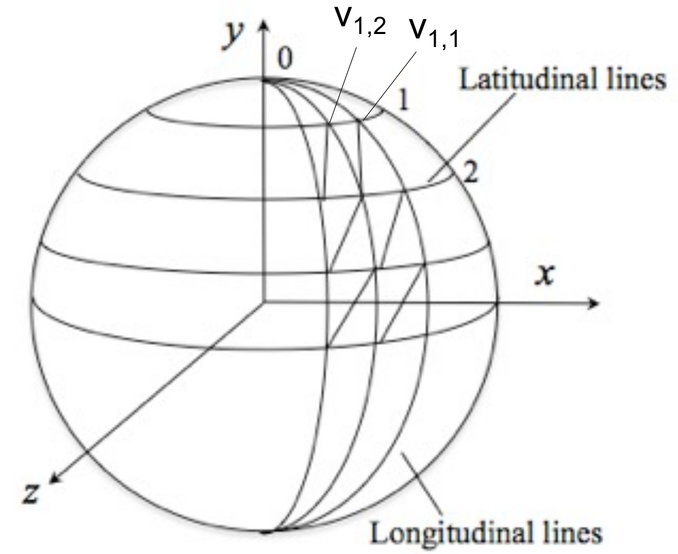
Sphere Tessellation

- To tessellate the the sphere, we divide the surface into m horizontal (latitudinal) strips and n vertical (longitudinal) strips, which means it has $m+1$ rows and $n+1$ columns (the last column repeats the first column) of vertices.



Cont'd

- By this division, we can compute the values of θ and φ for a vertex at ***i*th row and *j*th column**, $v_{i,j}$



$$\theta_{i,j} = i\pi/m \quad (i=0,1,\ldots,m)$$

$$\varphi_{i,j} = 2j\pi/n \quad (j=0,1,\ldots,n)$$

Calculate Coordinates

- Then coordinates of the vertex $v_{i,j}$ can be calculated by substituting θ with $i\pi/m$ and φ with $2j\pi/n$:

$$x = r \sin\theta \cos\varphi = r \sin(i\pi/m) \cos(2j\pi/n)$$

$$y = r \cos\theta = r \cos(i\pi/m)$$

$$z = r \sin\theta \sin\varphi = r \sin(i\pi/m) \sin(2j\pi/n)$$

- This calculation has to be done for every vertex

Generate Vertex Data Arrays

- Suppose we use `drawElements()` and **`gl.TRIANGLES`** to draw the sphere.
- Of course it can also be drawn using other methods
  - 1. `drawArrays()` and **`gl.TRIANGLES`** or
  - 2. `drawArrays()` or `drawElements()` and **`gl.TRIANGLE_STIRP`**, (you will need to arrange the vertices/indices in the correct order in the array)
- Use **`gl.TRIANGLES`** simplifies the the index generation process, even though it require a larger buffer space to store the vertex indices. For a simple object like sphere, this is not a big problem.

Vertex Coordinates

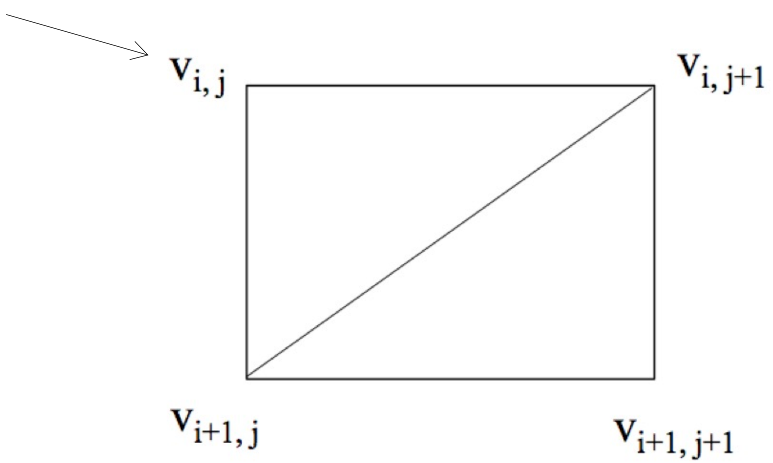
- We use two loops to calculate the vertex positions. Suppose we store the vertices in a JavaScript array:

```
var vertexPosition = [];
for (var i=0; i <= m; i++) {
   for (var j=0; j <= n; j++) {
      //Calculate x,y,z
      vertexPosition.push(x);
      vertexPosition.push(y);
      vertexPosition.push(z);
   }
}
```

- This procedure stores the vertex coordinates in such an order:
  - 1^{st} row of vertices (i=0 and j=0,1,2,…n), then
  - 2^{nd} row of vertices (i=1 and j=0,1,2,…n),
  - …

Vertex Indices

- Consider a vertex at ***i***th row and ***j***th column: 4 vertices $v_{i,j}$, $v_{i,j+1}$, $v_{i+1,j}$ and $v_{i+1,j+1}$ define two triangles:
  - $(v_{i,j}, v_{i+1,j}, v_{i,j+1})$ and
  - $(v_{i+1,j}, v_{i+1,j+1}, v_{i,j+1})$.



Cont'd

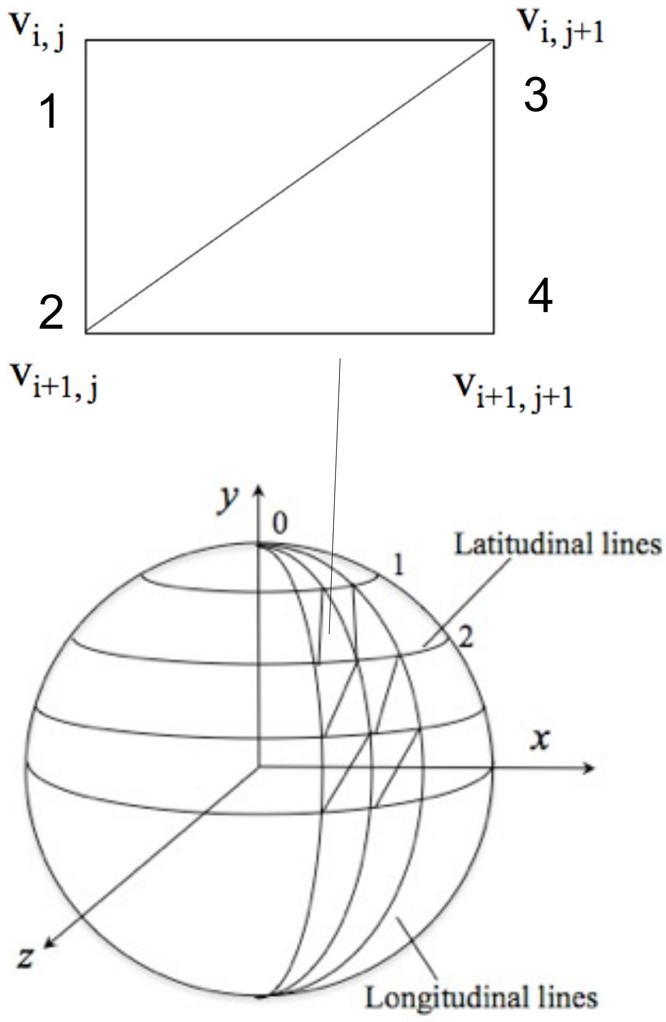
- According to the order of the vertex coordinates being stored, we can work out the indices of these vertices:

  index of $v_{i,j} = i*(n+1)+j$

  index of $v_{i,j+1}$ = index of $v_{i,j} + 1$

  index of $v_{i+1,j}$ = index of $v_{i,j} + n + 1$

  index of $v_{i+1,j+1}$ = index of $v_{i+1,j} + 1$



Cont’d

```
var indexData = [];
for (var i=0; i < m; i++) {
   for (var j=0; j < n; j++) {
        var v1 = i*(n+1) + j;//index of v_{i,j}
        var v2 = v1 + n + 1; //index of v_{i+1,j}
        var v3 = v1 + 1;     //index of v_{i,j+1}
        var v4 = v2 + 1;     //index of v_{i+1,j+1}
        // indices of first triangle
        indexData.push(v1);
        indexData.push(v2);
        indexData.push(v3);
        // indices of second triangle
        indexData.push(v3);
        indexData.push(v2);
        indexData.push(v4);
   }
}
```

Vertex Normals

- The vertex normal of a vertex, **n**, is simply

$$n_x = \sin\theta \cos\varphi = \sin(i\pi/m) \cos(2j\pi/n)$$
$$n_y = \cos\theta = \cos(i\pi/m)$$
$$n_z = \sin\theta \sin\varphi = \sin(i\pi/m) \sin(2j\pi/n)$$

- Notice that we have omitted the radius *r* from the formula of spheres, because we are working with unit vectors (length=1).

Cont’d

```
var normalData = [];
for (var i=0; i <= m; i++) {
   for (var j=0; j <= n; j++) {
      //Calculate nx,ny,nz
      normalData.push(nx);
      normalData.push(ny);
      normalData.push(nz);
   }
}
```

Further Reading

- Anyuru, A., WebGL Programming – Develop 3D Graphics for the Web
  - Chapter 3