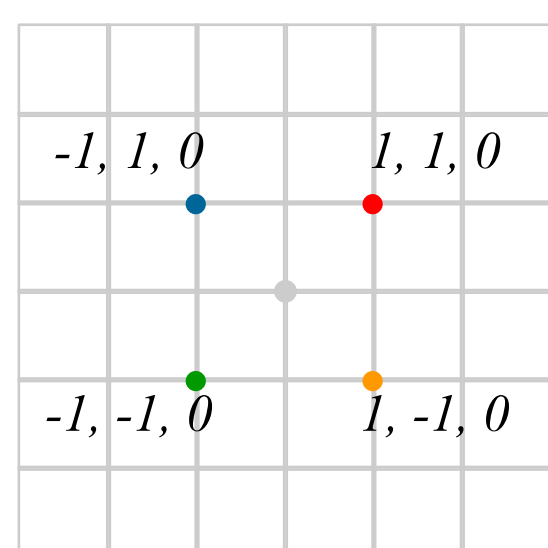
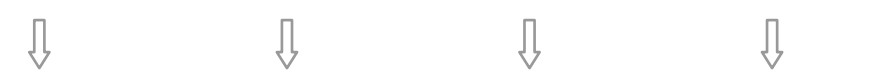
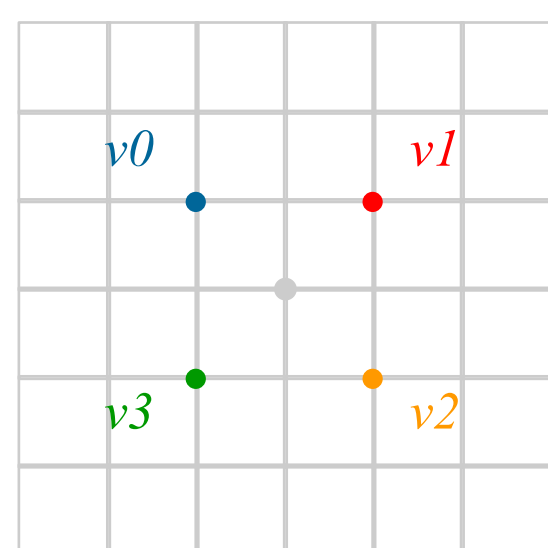
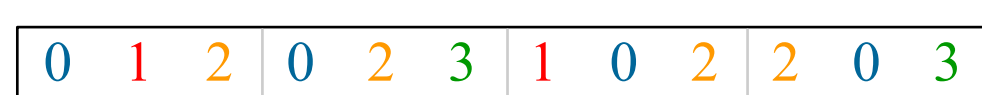
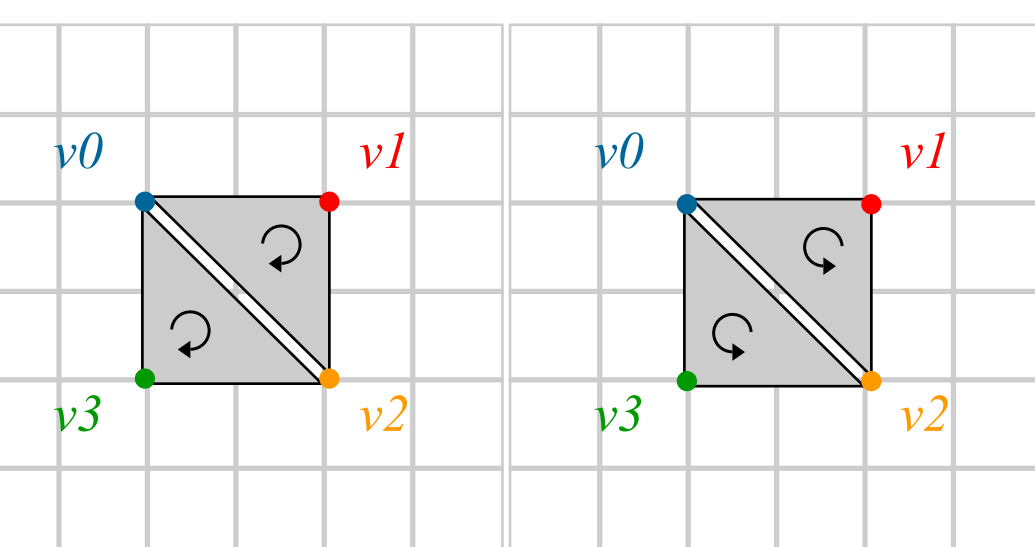
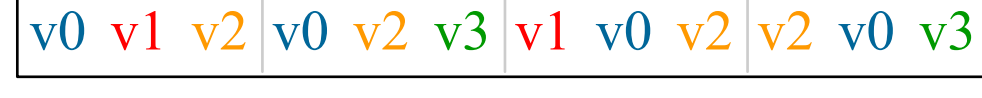
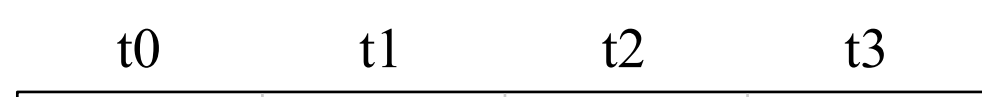


vertex buffer / index buffer

the vertex buffer is used to store the vertices coordinates. each vertex is stored as a triplet of XYZ values in a linear array.



now given this vertex buffer, the index buffer describes how the triangles of a mesh are built and stores triplets of vertices' indices.



the visibility of a triangle depends on its winding. the winding is the order in which the vertices of a triangle are associated ; CW or CCW.

- CW ClockWise
- CCW Counter ClockWise

given 4 vertices (points in 3D space)

the vertex buffer will store them like this

so given the following coordinates

here's what our vertex buffer looks like

if we want to build a double sided quadrilateral, we'll need these triangles

we'll associate the vertices' position in the vertex buffer. V0 is our first vertex in the vertex buffer ; it's index is 0. V1 is index 1, V2 is index 2 and V3 is index 3.

- some benefits of using separate buffers:
- it saves memory because we re-use vertices
 - therefore it saves data transmission time
 - the topology (point connectivity) is independant from the shape (point locations)
 - it makes model deformation easy

vector

length

$$\text{length} = \sqrt{x^2 + y^2 + z^2}$$

normalization

$$\begin{aligned} x / \text{length} \\ y / \text{length} \\ z / \text{length} \end{aligned}$$

scalar multiplication

$$\begin{aligned} x * \text{scale} \\ y * \text{scale} \\ z * \text{scale} \end{aligned}$$

addition

$$\begin{aligned} x0 + x1 \\ y0 + y1 \\ z0 + z1 \end{aligned}$$

subtraction

$$\begin{aligned} x0 - x1 \\ y0 - y1 \\ z0 - z1 \end{aligned}$$

cross product

$$\begin{aligned} y0 * z1 - z0 * y1 \\ z0 * x1 - x0 * z1 \\ x0 * y1 - y0 * x1 \end{aligned}$$

dot product

$$x0 * x1 + y0 * y1 + z0 * z1$$

angle

$$\text{dp} = v0.\text{dotProduct}(v1)$$

$$\text{acos}(\text{dp} / (v0.\text{length} * v1.\text{length}));$$

matrix

matrix

$$\begin{bmatrix} m0 & m1 & m2 & m3 \\ m4 & m5 & m6 & m7 \\ m8 & m9 & m10 & m11 \\ m12 & m13 & m14 & m15 \end{bmatrix}$$

identity (no transform)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

transpose

$$\begin{bmatrix} m0 & m4 & m8 & m12 \\ m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \end{bmatrix}$$

multiplication

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$\begin{aligned} A &= a * 0 + b * 2 + c * 4 + d * 6 \\ B &= a * 1 + b * 3 + c * 5 + d * 7 \\ C &= e * 0 + f * 2 + g * 4 + h * 6 \\ D &= e * 1 + f * 3 + g * 5 + h * 7 \end{aligned}$$

transform point 3d (multiplication again)

$$\begin{bmatrix} x & y & z & w \end{bmatrix} * \begin{bmatrix} m0 & m1 & m2 & m3 \\ m4 & m5 & m6 & m7 \\ m8 & m9 & m10 & m11 \\ m12 & m13 & m14 & m15 \end{bmatrix}$$

$$\begin{aligned} x' &= x * m0 + y * m4 + z * m8 + w * m12 \\ y' &= x * m1 + y * m5 + z * m9 + w * m13 \\ z' &= x * m2 + y * m6 + z * m10 + w * m14 \\ w' &= x * m3 + y * m7 + z * m11 + w * m15 \end{aligned}$$

scale

$$\begin{bmatrix} *sx & 0 & 0 & 0 \\ 0 & *sy & 0 & 0 \\ 0 & 0 & *sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} m0 &*= sx \\ m5 &*= sy \\ m10 &*= sz \end{aligned}$$

rotation X

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cx & -sx & 0 \\ 0 & sx & cx & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation Y

$$\begin{bmatrix} cy & 0 & sy & 0 \\ 0 & 1 & 0 & 0 \\ -sy & 0 & cy & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation Z

$$\begin{bmatrix} cz & -sz & 0 & 0 \\ sz & cz & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ +tx & +ty & +tz & 1 \end{bmatrix}$$

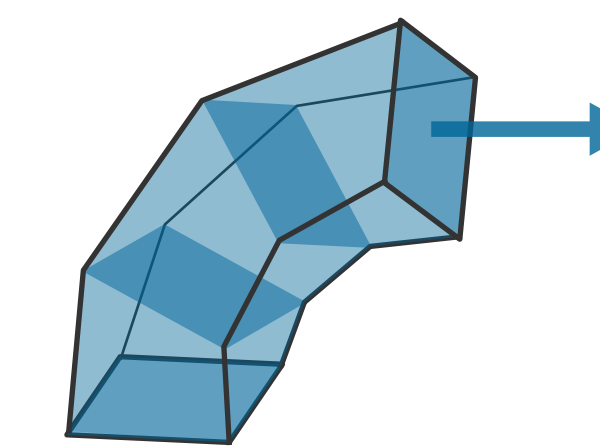
$$\begin{aligned} m12 &+= tx \\ m13 &+= ty \\ m14 &+= tz \end{aligned}$$

projection

$$\begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & -(f+n) & -(2fn) \\ 0 & 0 & /f-n & /f-n \end{bmatrix}$$

n, f = distances to near, far planes
e = focal length = 1 / tan(FOV / 2)
a = viewport height / width

linear extrusion

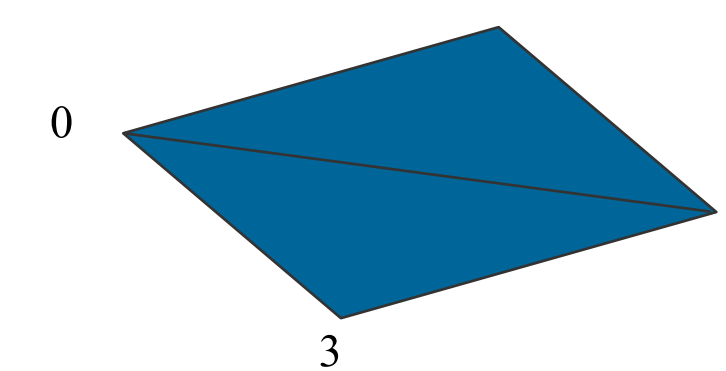


if we need o keep the caps:
for the top 4-5-6 & 4-6-7 are the same as the original 0-1-2 & 0-2-3 triangles simply add 4 (the count of new vertices) to the original indices
0+4=4 1+4=5 2+4=6 => 4-5-6
0+4=4 2+4=6 3+4=7 => 4-6-7
if the vertices describe a simple polygon, it will work for all the original faces.

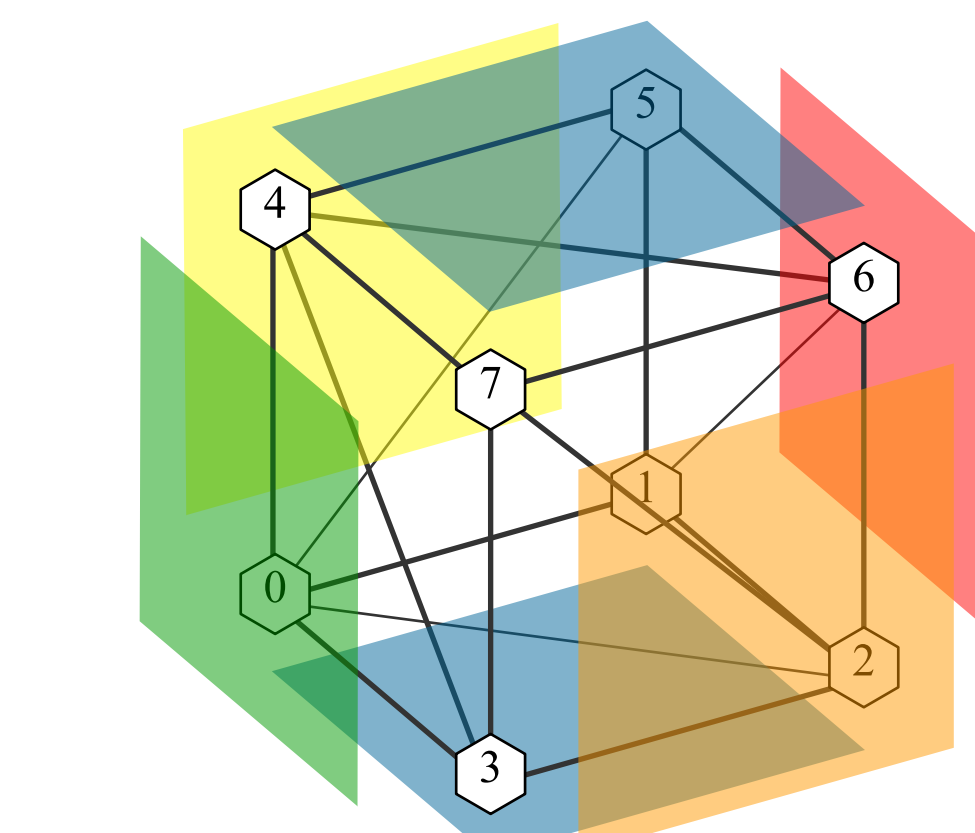
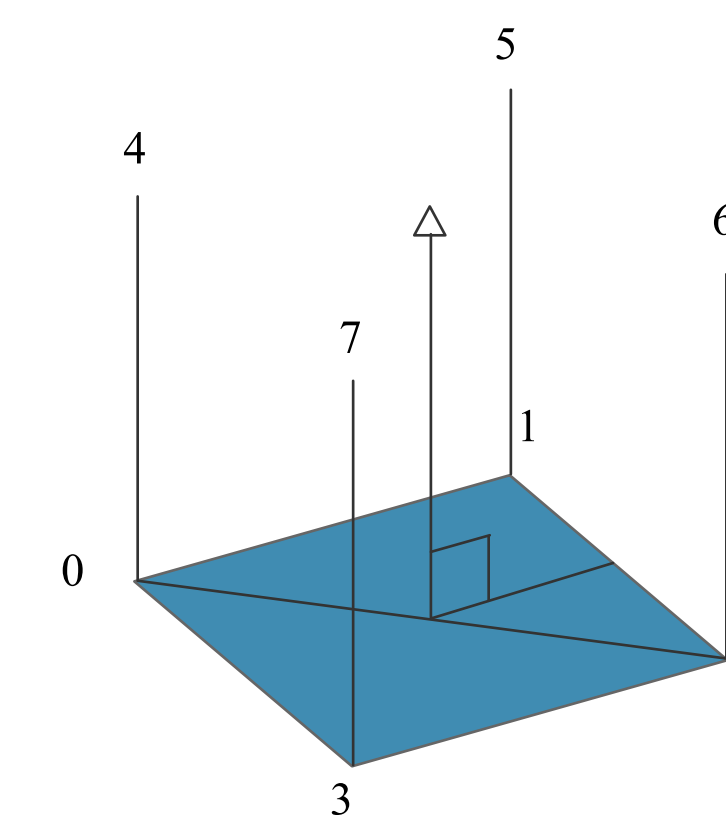
for the bottom cap, flip the original indices to make them point outwards.
0-1-2 => 1-0-2
0-2-3 => 2-0-3

then create the necessary indices:
0-4-5, 0-5-1, 1-5-6, 1-6-2, 2-6-7, 2-7-3, 3-7-4, 3-4-0

starting from a given set of vertices



create new vertices and push them into the vertex buffer



very complex and organic looking shapes can be achieved by performing multiple linear extrusions. it's usually called a LOFT and it's one of the most useful tools in the 3D toolbox.

face subdivision

