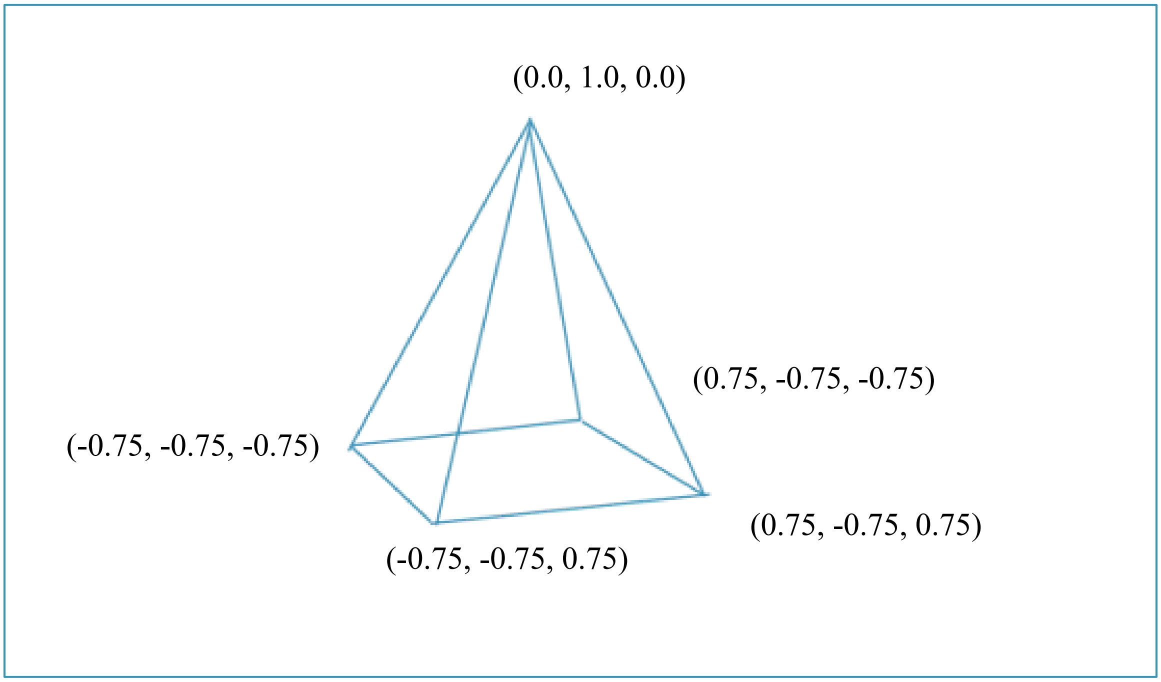
Tutorial 1 Week 2

1. Draw a pyramid using the following information



2. Drawing a disc:

Take a look at the picture:Point 'P (x, y, z)' can be defined as:

P.x = Pcos(theta);
P.y = Psin(theta)
P.z = 0;

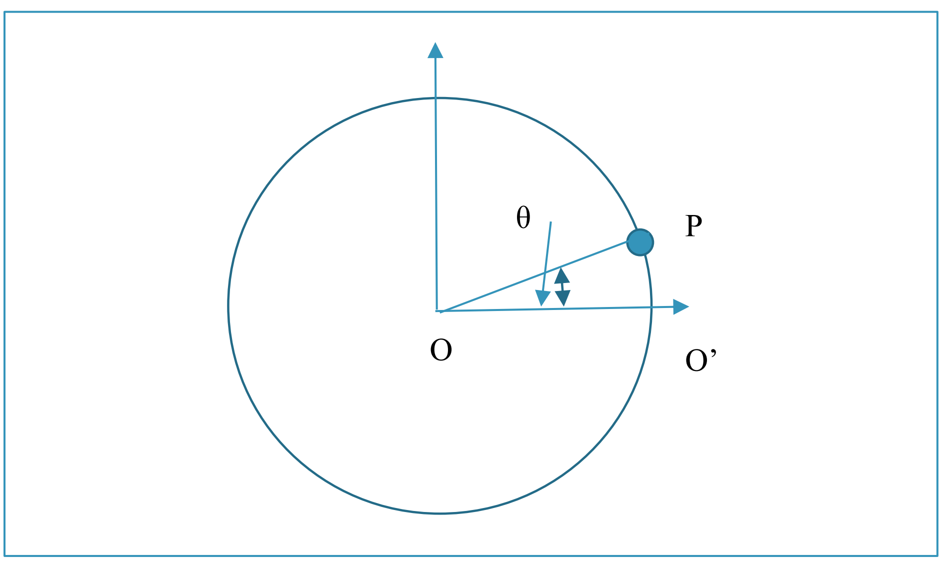


Figure 1: Drawing a disc

Now, OO'P forms a triangle.

So you need to traverse from 0 degree to 360 degree in XY plane to get all the vertices of a circle or disc Here is the pseudo code:

```
for (i = 0; i < NumPoints; ++i) {

              theta = i*(angle_interval)*PI / 180.0f;

              points[index][0] = cos(theta);
              points[index][1] = sin(theta);
              points[index][2] = 0.0;
              points[index][3] = 1.0;

              :
              :
}
```

As you complete the traversal at a certain interval (here, angle_interval), you will find the following triangles. You also need to consider the center of the disc. So the total number of points or vertices is (NumPoints+1).

Count the number of vertices, number of indices that form the triangles to construct the surface

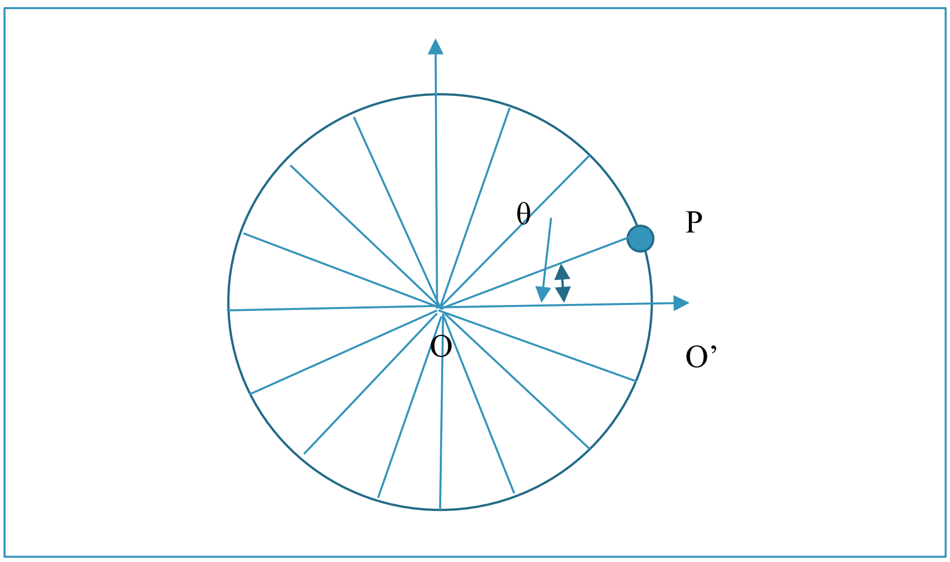


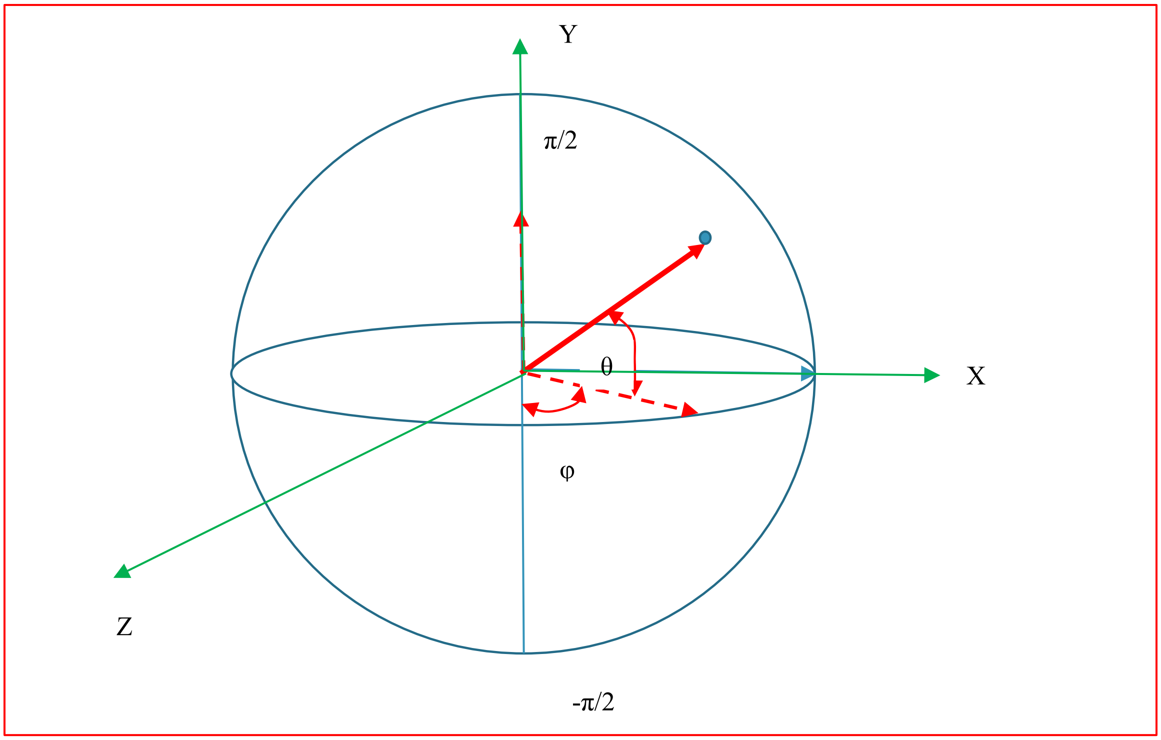
Figure 2: Calculating the vertex points of a cone or disc.

3. Drawing a cone:

Hint:

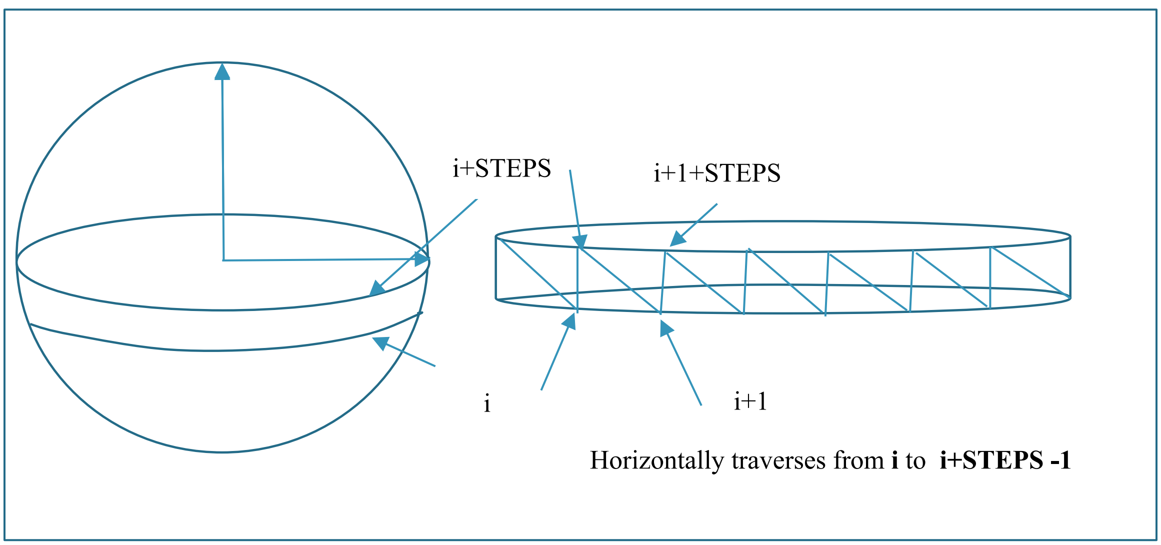
'Cone' is very similar to disc but the center is not in the same plane. In order to draw a cone you need to draw a disc in XZ plane first with center along the y-axis. The number of vertices, number of triangles will be the same as that of a disc.

4. Drawing a sphere:



Constructing the surface:

`Take a look at the following picture?`

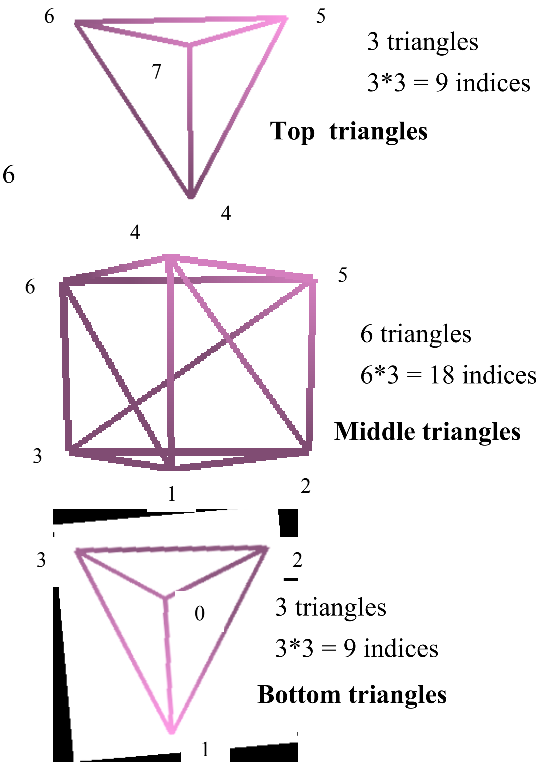


STEP = 3

Number of Vertices = 8 (0 to 7)

Number of Triangles = 3 + 6 + 3 = 12

Number of Indices = Number of Triangles*3 = 36



STEP = 4

Number of Vertices = 14 (0 to 13)

Number of Triangles = 4 + 16 + 4 = 24

Number of Indices = Number of Triangles*3 = 72

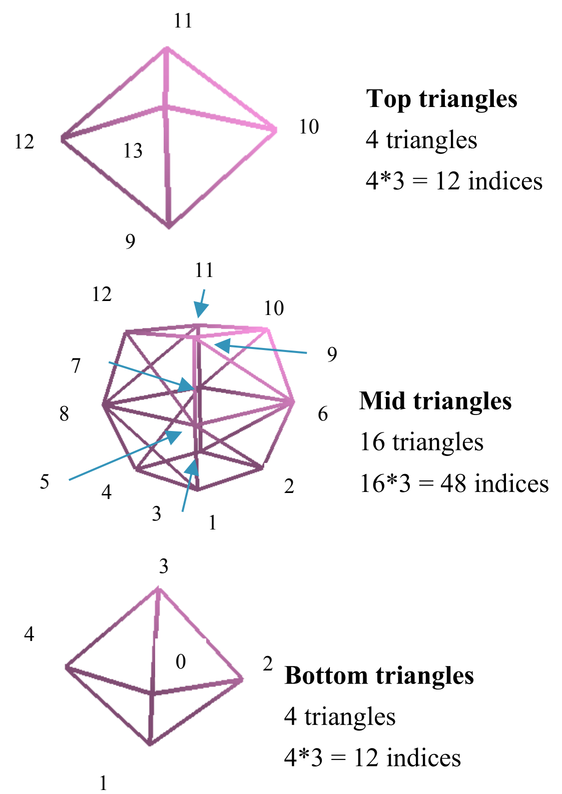


Figure 2: (Top) Polygon with STEP = 3 and (bottom) Polygon with STEP = 4.

Now, after observing the above-mentioned two models at STEP = 3 and STEP = 4, you can generalize the number of vertices, the number of triangles and the number of indices as follows (when there are no redundant vertices):

Number of vertices = STEP*(STEP-1) + 2

Number of triangles = STEP*(STEP-1)*2

Number of indices = Number of triangles*3 = STEP*(STEP-1)*6

```
for (double b = -STEP /2.0; b <= STEP/2.0; b++) {
    if ( b == -STEP/2.0 || b == STEP/2.0){
        theta = (1.0 * b / STEP) *kPI;
         vertex[i].x = 0;
              vertex[i].y = sin(theta);
              vertex[i].z = 0;
              vertex[i].w = 1.0f;

    }
    else{
         for (int a = 0; a < STEP; a++) {
              // lat/lon coordinates
              phi = (1.0 * a / STEP) * 2 * kPI;
              theta = (1.0 * b / STEP) *kPI;

              vertex[i].x = cos(theta)*sin(phi);
              vertex[i].y = sin(theta);
              vertex[i].z = cos(theta)*cos(phi);
              vertex[i].w = 1.0f;
              i++ ;
        }
     }
    }
```

Top/ bottom points

Middle vertices

Now write the code that constructs the surface as follows:

For top and bottom surface (as shown in Figure 2):

Top and bottom vertices form triangles from three vertice-sets (instead of rectangle that needs to divided into two triangles as needed for other two consecutive rings)) with the previous (top) or next (bottom) ring as shown in Figure 2. Number of triangles (at the top and bottom) is equal to the number of STEP.

Pseudo code for constructing bottom surface:

```
for (int i = 0; i < (STEP); i++) {

        if (i == STEP-1 ) // does the wrap around {
               Write your code
        }
        else  // construct a triangle in each iteration
        {
              Write your code
        }
}
```

Pseudo code for top surface:

```
for (int i = NUMVERTICES-STEP-1; i < NUMVERTICES-1; i++) {

              if (i == NUMVERTICES - 2) { // does the wrap around part
                     //write your code
              }
              else
              { // construct a triangle in each iteration

                   // write your code
              }
}
```

For middle surface:

There are STEP – 1 rings (we are not considering the top and bottom points) , each two consecutive rings form a strip. As we are not considering the redundant vertices in each ring (as happens when in vertex loop, a <= STEP as we did during the tutorial), so when the index reaches the last index in the ring, it needs to be wrapped around with the first vertex of the ring.The difference between two corresponding vertices in two consecutive rings is STEP. For a particular ring, if the first vertex is **i,** the last vertex is **i+STEP-1**;

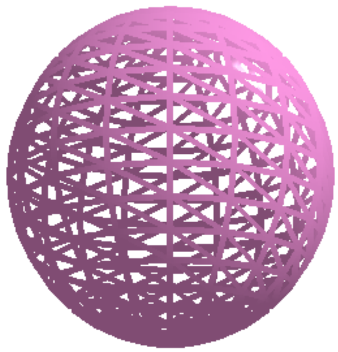
Observe Figure 1 and Figure 2, write the code for the middle surface as follows:

```
for (int i = 1; i < (NUMVERTICES - STEP - 1); i += STEP) {

     for (int j = i; j < (i + STEP); j++) {

         if (j == i + STEP - 1){// does wrap up
             // each iteration construct two triangles
         }
         else {
             // construct the rest of the trianglulated surface
         }
    }
}
```

Once done, increase the number of STEP (18, 36 etc.) to get a smooth surface as follows.



5. Use of different types of projection

Now let us consider different types of projections. Till now, we only considered orthogonal projection. How will you consider perspective projection?

Use the following view matrix:

```
view = glm::lookAt(vec3(0.0f, 0.0f, 6.0f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
```

Next try the following projection matrix one by one:

1. `projection = glm::ortho(-4.0f, 4.0f, -4.0f, 4.0f, 4.5f, 100.0f);`
2. `projection = glm::perspective(radians(70.0f), aspect, 4.5f, 100.0f);`
3. `projection = glm::frustum(-4.0f, 4.0f, -4.0f, 4.0f, 4.5f, 100.0f);`