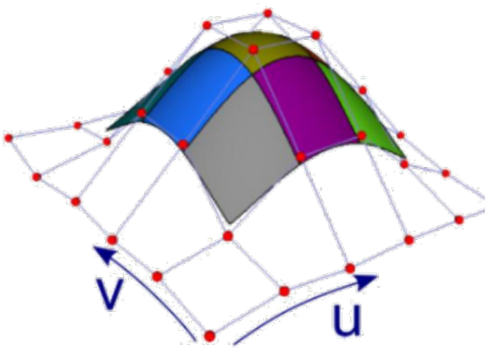
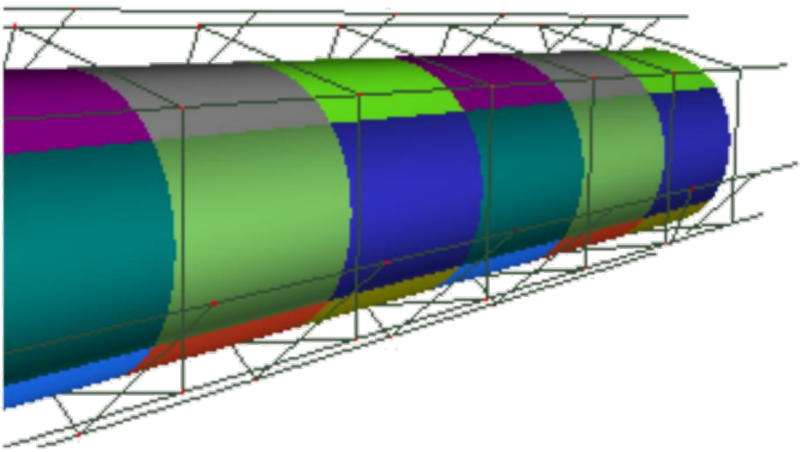
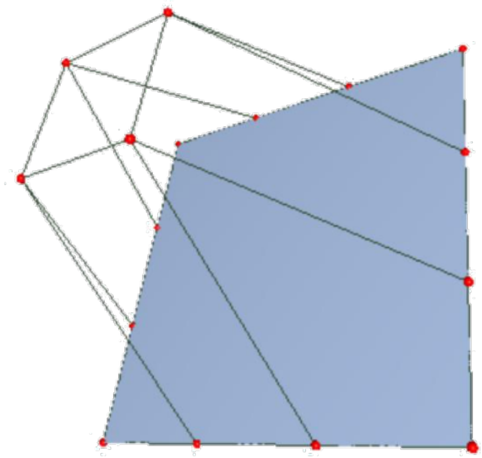
TP Curves and Parametric Surfaces





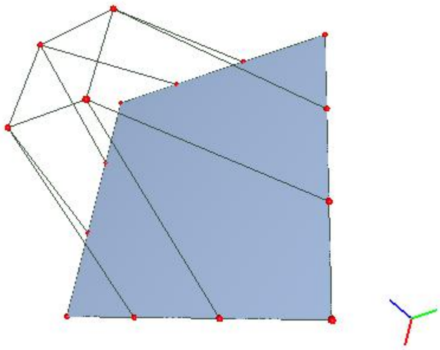


MASSON Margaux & MARCAIS Thibault

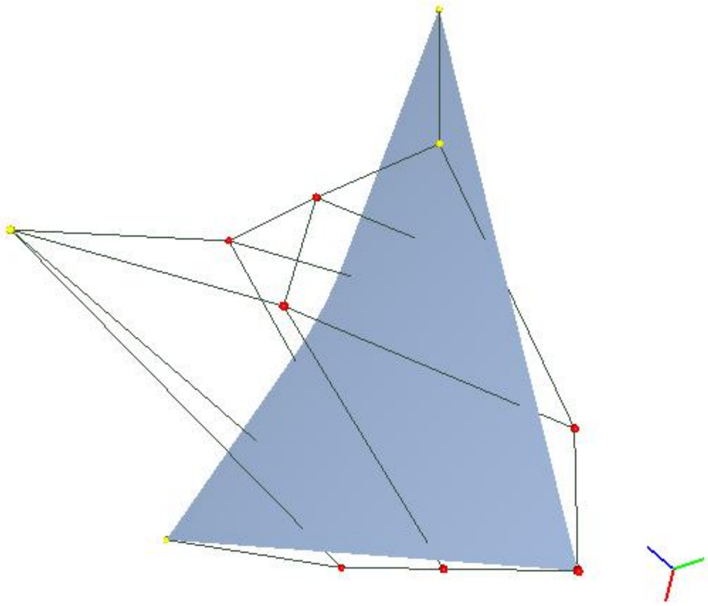
4ETI

I) Current operation of the program

When we run the supplied program, we get the following 3D scene:



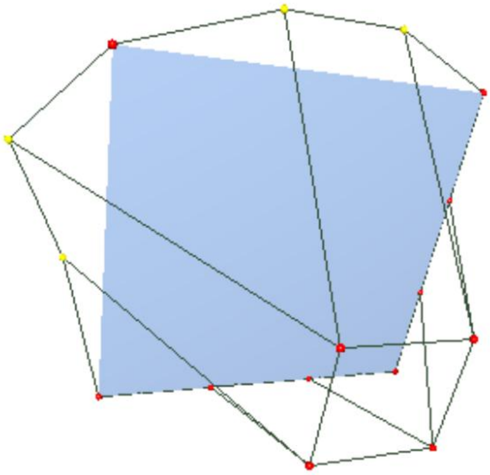
=> So we have an editable plane surface, indeed we can deform this surface by selecting the vertices (Ctrl + click) and moving them as follows:

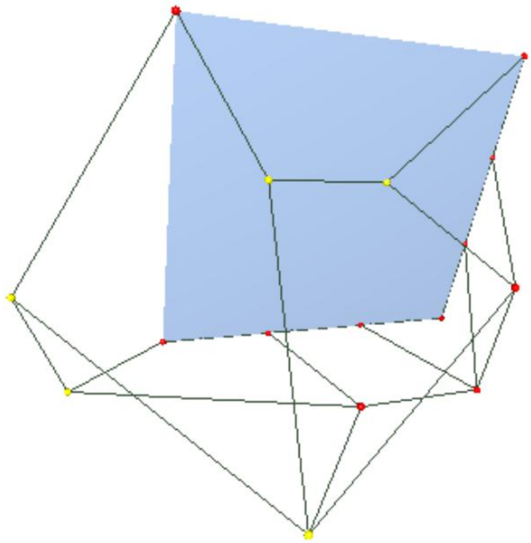


We are now studying the code of the spline_evaluator.cpp file. It is in this file that the surface observed previously is implemented. It is coded directly by indicating the vertices of the latter in the parentheses operator function :

```
float spline_evaluator::operator()(float const u,float const v) const
{
    float S = P(0,0)*(1-u)*(1-v) +
              P(0,3)*(1-u)*   v +
              P(3,0)*   u *(1-v) +
              P(3,3)*   u *   v ;
    return S;
}
```

In order to deform this surface, we must move the vertices P_{00}, P_{03}, P_{30} or P_{33} that is, the corners of the square (initial surface).
We can check this in the situations below:

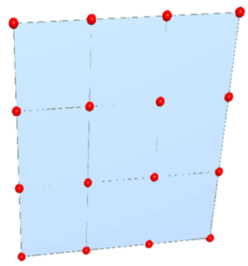




=> We note that when we move the points other than the vertices of the square, the surface is not modified.

II) Bézier tiles

A Bezier curve is a polynomial curve segment of degree n defined by n + 1 points called control points. The matrix P used for the x, y and z coordinates is noted:



$$P = \begin{pmatrix} (0,0,0) & \left(0,\frac{1}{3},0\right) & \left(0,\frac{1}{3},0\right) & (0,1,0) \\ \left(\frac{1}{3},0,0\right) & \left(\frac{1}{3},\frac{1}{3},0.7\right) & \left(\frac{1}{3},\frac{2}{3},0.7\right) & \left(\frac{1}{3},1,0\right) \\ \left(\frac{2}{3},0,0\right) & \left(\frac{2}{3},\frac{1}{3},0.7\right) & \left(\frac{2}{3},\frac{2}{3},0.7\right) & \left(\frac{2}{3},1,0\right) \\ (1,0,0) & \left(1,\frac{1}{3},0\right) & \left(1,\frac{2}{3},0\right) & (1,1,0) \end{pmatrix}$$

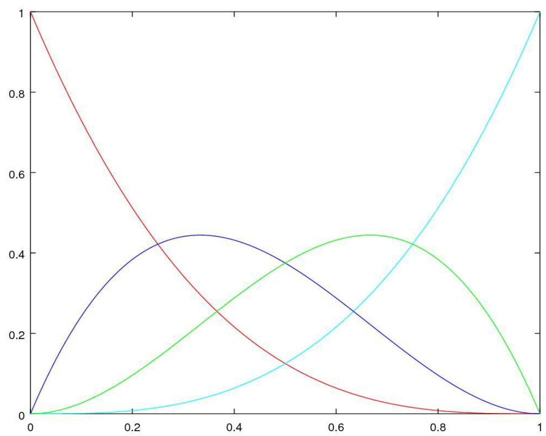
The 16 control points are evenly distributed, the points that are not at the edges have coordinates z = 0.7. The curves are therefore polynomials of degree 3 because we have 4 control points for each case.

The calculation of Bézier's surface is: $S(u,v) = U^t \, \mathrm{M}^t \, \mathrm{P} \, \mathrm{M} \, V$

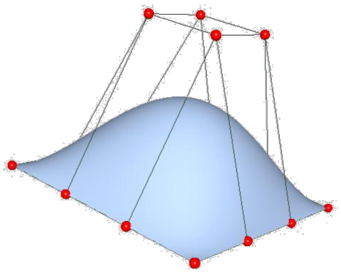
Each term is explained as:

$$V = \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix} \qquad U = \begin{pmatrix} u^3 \\ u^2 \\ u \\ 1 \end{pmatrix} \qquad M = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The matrix M defines the weights of the points at each of the coordinates.



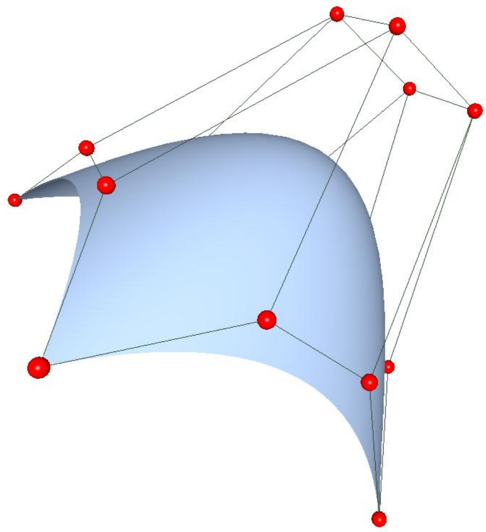
Weighting function for 4 control points



The Bezier surface passes well through the edge control points as opposed to Spline surfaces. Changing a control point causes a complete change of the surface. The Bézier surface is tangent to the P (1,0) P (1,1) segment at points A (1,0,0).

The diff_u and diff_v derivatives allow the calculation of the normal at each point. These operators are described as the following matrix calculations:

- compared to u : dSu = dU*M*P*M*V
- compared to v : dSv = U*M*P*M*dV



The mathematical formula for defining the norm in u and v is :

$$n=\langle dSu \mid dSv \rangle$$

A normalization operation is needed after this scalar product.

III) Spline Tiles

After using the Bezier surface, we want to implement a Spline surface to have multiple tiles linked to each other keeping the impression of a smooth overall surface.

The calculation of the surface is the same as for that of the Bézier functions, that is to:

$$S(u,v) = U^t.M^t.P.M.V$$

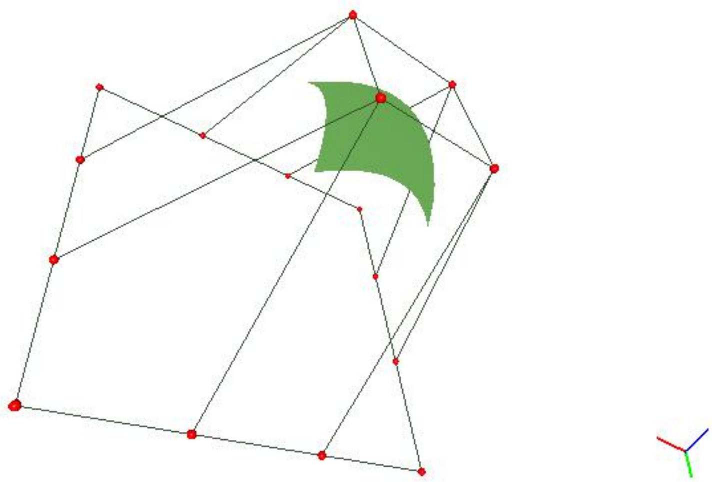
However, the matrix M is now written :

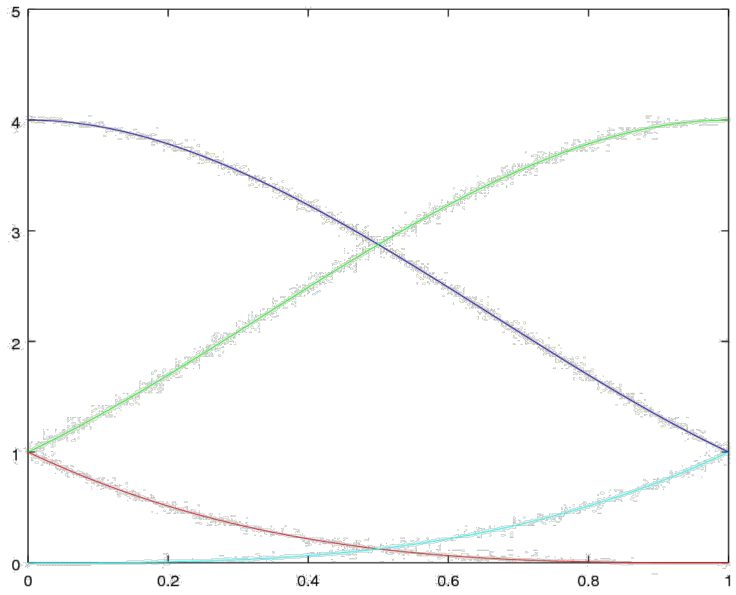
$$M = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

We now have the following code:

```
float spline_evaluator::operator()(float const u,float const v) const
{
   mat4 M(-1,3,-3 ,1,
           3,-6 ,0, 4,
          -3, 3, 3,1,
           1,0,0,0); //B-spline
    M=M/6;

    mat1x4 U(pow(u,3),pow(u,2),u,1);
    mat4x1 V(pow(v,3),pow(v,2),v,1);

    float S=U*transposed(M)*P*M*V;
    return S;
}
```

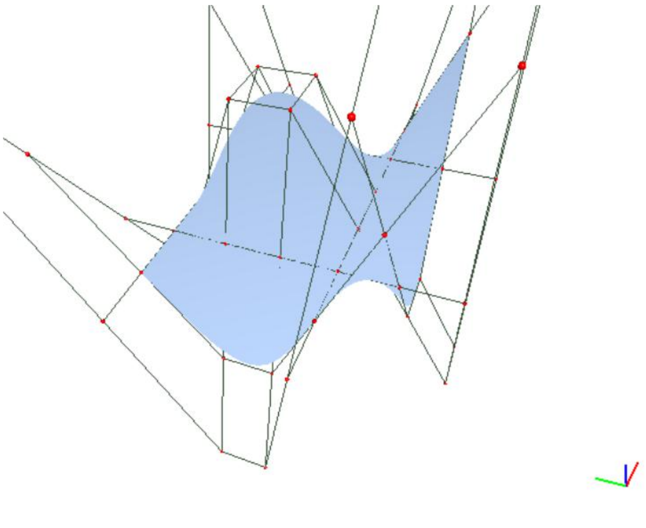
We obtain the following B-Spline surface (smooth, locally controllable and of expected shape):

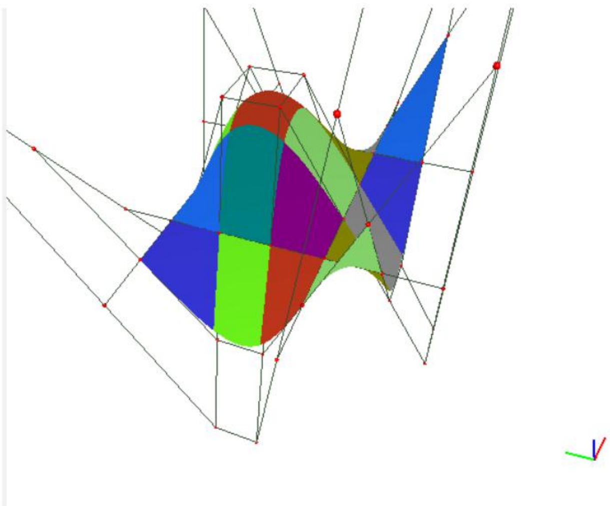




Weighting function for 4 control points

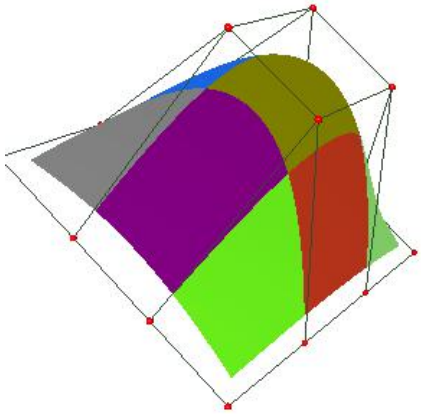
By using the Control Polygon menu that allows us to add or remove rows / columns from the control polygon, we get several tiles forming a smooth surface:



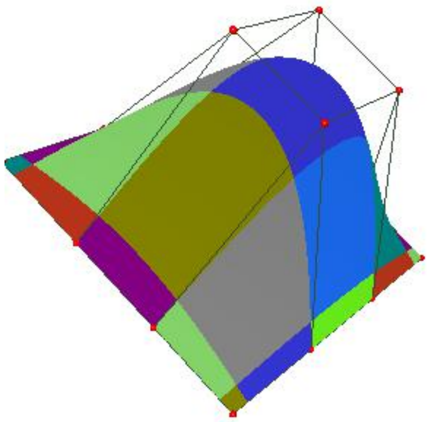


IV) Extensions to various forms

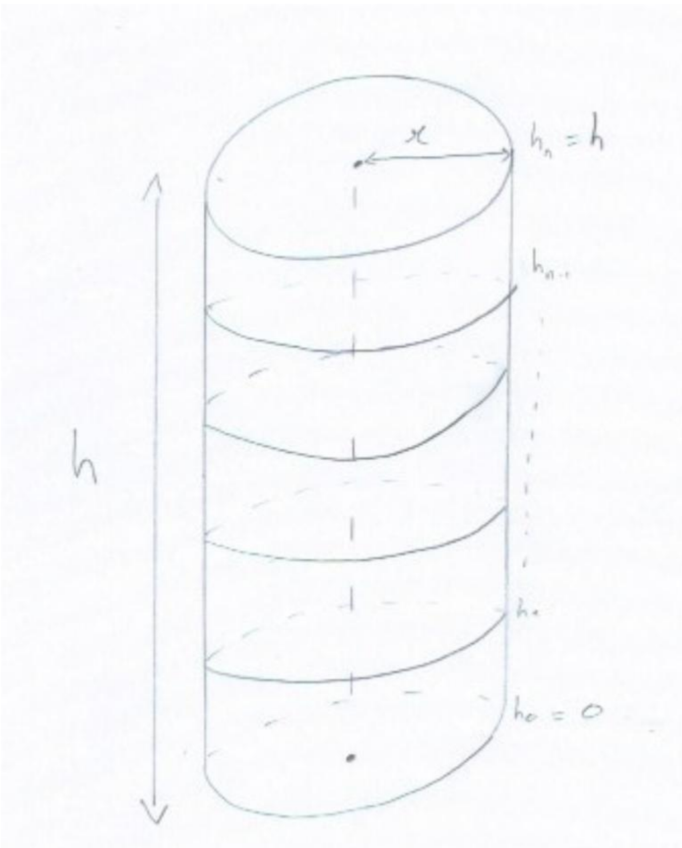
When we duplicate the vertices of the control polynomial we get this figure:

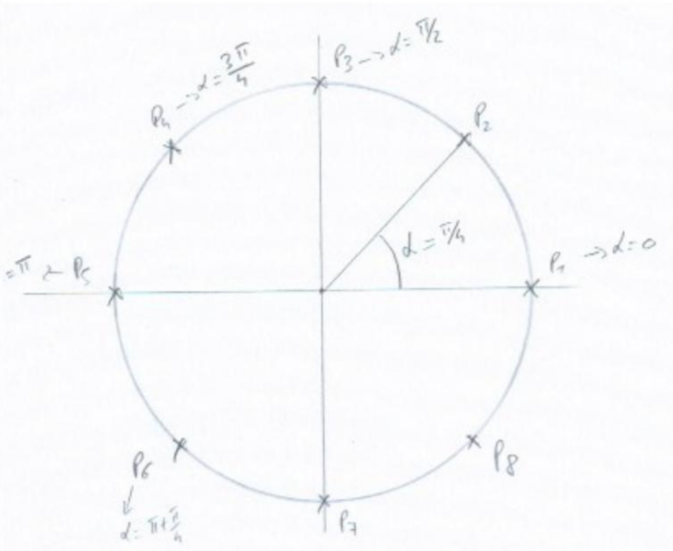


Then a second time :



We now want to implement other surfaces. We start by modeling a cylinder following the following reasoning:





We will first reason by "layers" (or "stages" of h) to create the cylinder end by end. We decide to place 8 points on each floor $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ *and* P_8 as in the diagram, so we will place them according to α that we will increment by $\pi/4$ to cover the circle and position the points as desired. Then, we go from floor to floor until we reach the desired height h by incrementing h each time the 8 points are well placed. Which gives us the coordinates (x, y, z) next:

$x = r * cos(2\pi\alpha)$

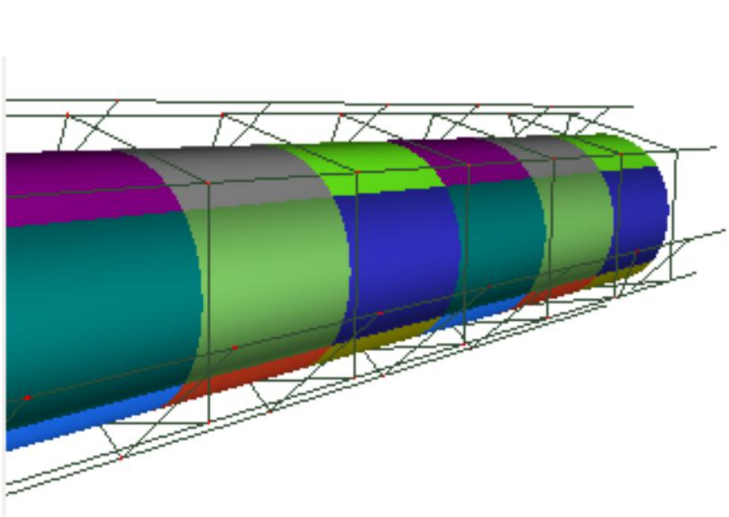
$y = r * sin(2\pi\alpha)$

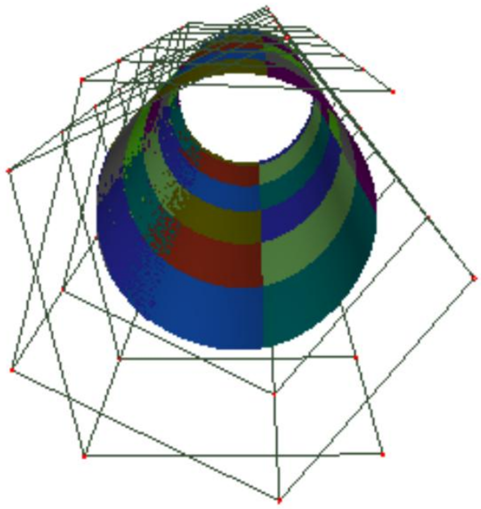
$z = height\ h$

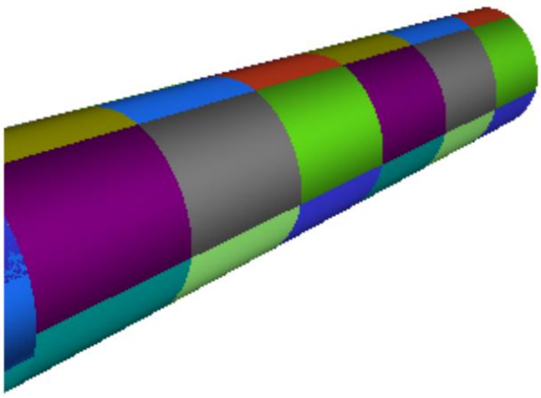
Which brings us to the following code in the polygon_grid.cpp file:

```
polygon_grid::polygon_grid()
{
    size_u_data=9;
    size_v_data=9;
    float h=0;
    float r=1;
    float alpha=0;

    for(int i=0;i<size_v_data;i++)
    {
      for(int j=0;j<size_u_data;j++)
        {
            vec3 v3={static_cast <float>(r*cos(2*3.14*alpha)),static_cast <float>(r*sin(2*3.14*alpha)),h};
            vertex_data.push_back(v3);
            alpha=alpha+3.14/4;
        }
        h++; //on passe a l etage suivant
        alpha=0; //on reinitialise l angle alpha pour bien positionner les points

      }
 }
```

We now have the following cylinder:







Then, to model a sphere, we use the following equations:

$$x = r * \cos(\theta) * \cos(\varphi)$$
$$y = r * \cos(\theta) * \sin(\varphi)$$
$$z = r * \sin(\theta)$$

Which brings us to the following code:

```
polygon_grid::polygon_grid()
{
     size_u_data=16;
     size_v_data=16;
     float Nu=size_u_data;
     float Nv=size_v_data;
      float r=2;

     for(int i=0;i<Nu;i++)
     {
            for(int j=0;j<Nv;j++)
         {
              float const phi = -2*M_PI +static_cast <float>(i)/(Nu-1)*4*M_PI;
              float const theta = -M_PI +static_cast <float>(j)/(Nv-1)*2*M_PI;
              vec3 v3={static_cast <float>(r*cos(theta)*cos(phi)),static_cast
<float>(r*cos(theta)*sin(phi)),static_cast <float>(r*sin(theta))};
vertex_data.push_back(v3);

         }

    }
```

We obtain the following sphere: