Finally, if either MAP2_VERTEX_3 or MAP2_VERTEX_4 is enabled, then the normal to the surface is computed. Analytic computation, which sometimes yields normals of length zero is one method which may be used. If automatic normal generation is enabled, then this computed normal is used as the normal associated with a generated vertex. Automatic normal generation is controlled with **Enable** and **Disable** with symbolic the constant AUTO_NORMAL. If automatic normal generation is disabled, then a corresponding normal map, if enabled, is used to produce a normal. If neither automatic normal generation nor a normal map are enabled, then no normal is sent with a vertex resulting from an evaluation (the effect is that the current normal is used).

For MAP_VERTEX_3, let $\mathbf{q} = \mathbf{p}$.

For MAP_VERTEX_4, let $\mathbf{q} = (x/w, y/w, z/w)$,

where $(x, y, z, w) = \mathbf{p}$. Then let

$$\mathbf{m} = \frac{\partial \mathbf{q}}{\partial u} \times \frac{\partial \mathbf{q}}{\partial v}.$$

Then the generated analytic normal, $\mathbf{n}$, is given by $\mathbf{n} = \mathbf{m}/\|\mathbf{m}\|$.

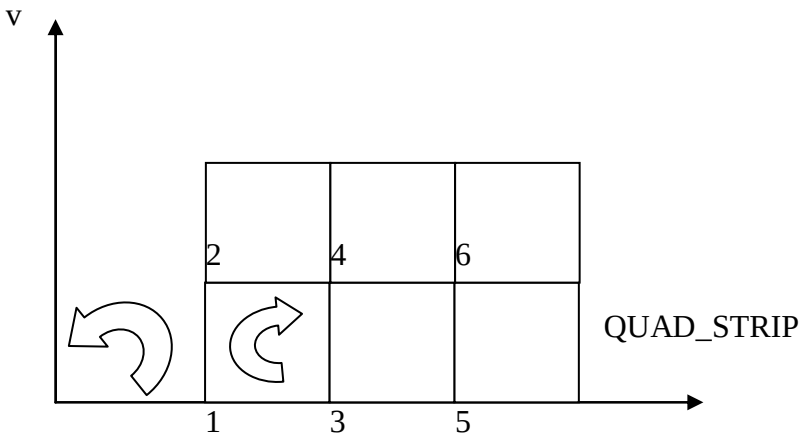
Pour les surfaces de Bezier :

La surface est dessinée avec les instructions suivantes :

```
  glFrontFace(GL_CW);  // Pour que la face avant soit tournée vers le haut
  // Pour séparer le calculs des faces avant et arrière
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
  glPushMatrix();
    // Enable the evaluator
    glEnable(GL_MAP2_VERTEX_3);

    // Générer automatiquement les normales pour la surface évaluée.
    // n = dp/du x dp/dv
    // Attention :
    // Si GL_LIGHT_MODEL_TWO_SIDE est true, il faut que la normale à
    // la face avant pointe dans la direction u x v.
    // Par exemple, ici, il faut que la face avant soit vers le haut
    // puisque u x v pointe vers le haut.
    glEnable(GL_AUTO_NORMAL);
    // Evaluate the grid
    glEvalMesh2(GL_FILL, 0, 10, 0, 4);

//*********************************************************************
//     glEvalMesh2(GL_FILL, 0, 10, 0, 4); est équivalent à :
//*********************************************************************
/*    int nu = 10;
    int nv = 4;
    GLfloat du = 1.0f / nu;
    GLfloat dv = 1.0f / nv;
    for (int j = 0;  j < nv; j += 1)
    {
```

```
        glBegin(GL_QUAD_STRIP);
        for (int i = 0; i <= nu; i += 1)
        {
           glEvalCoord2f(i*du, j*dv);
           glEvalCoord2f(i*du, (j+1)*dv);
        }
        glEnd();
    }*/
//*********************************************************************

    glDisable(GL_AUTO_NORMAL);
    glDisable(GL_MAP2_VERTEX_3);
      glPopMatrix();
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
  glFrontFace(GL_CCW);
```

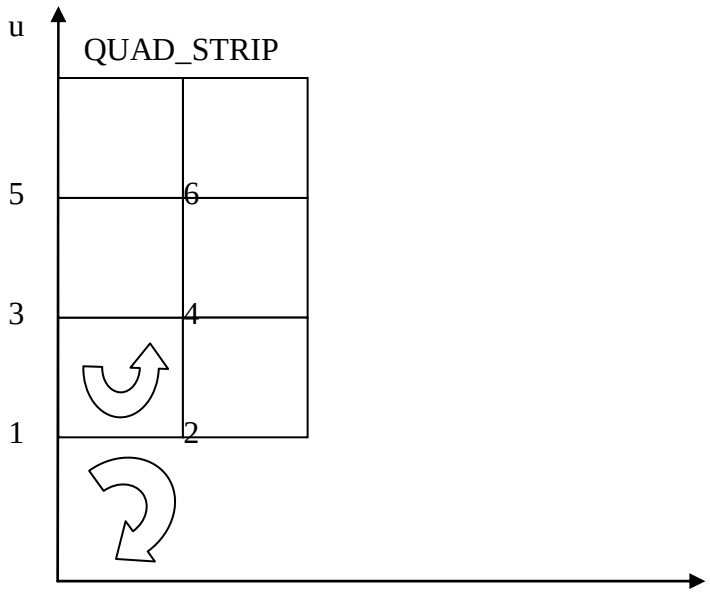
On a donc le schéma suivant :



(u x v) est dirigé vers nous.

Les QUAD_STRIP sont dessinés le long de l'axe u dans le sens CW.
Donc, pour que les faces avant soient dirigées vers nous (comme la normale), il faut appeler la fonction : glFrontFace(GL_CW).

Si les axes u et v sont inversés, on obtient la même chose, mais dans le sens contraire.



(u x v) est dirigé vers l'arrière.

En résumé, si (u x v) est dans le sens CCW, les faces sont dans le sens CW, et vice versa.
Donc, si on veut que les lumières soient correctes, il faut toujours appeler la fonction : glFrontFace(GL_CW).

Si on veut que les faces avant soient vers le haut, par exemple, il ne faut pas jouer avec la fonction glFrontFace(GL_CW), mais plutôt avec la position des offsets dans la fonction :

```
glMap2f(GL_MAP2_VERTEX_3,      // Type of data generated
    0.0f,               // Lower u range
    1.0f,              // Upper u range
    3*nNumPointsDir_v, // Distance between points in u direction
    nNumPointsDir_u,   // Dimension in u direction (order)
    0.0f,               // Lover v range
    1.0f,              // Upper v range
    3,                 // Distance between points in v direction
    nNumPointsDir_v,   // Dimension in v direction (order)
    &ctrlPoints[0][0][0]);  // array of control points
```

En inversant les valeurs en gras, on inverse les faces avant et arrière tout en gardant la lumière correcte (si on a appelé la fonction : glFrontFace(GL_CW)). Attention, si le Nb de points en u est différent du Nb de points en v, il faut aussi inverser **nNumPointsDir_u** et **nNumPointsDir_v !**

Pour les NURBS, il faut toujours appeler la fonction glFrontFace(**GL_CCW**).