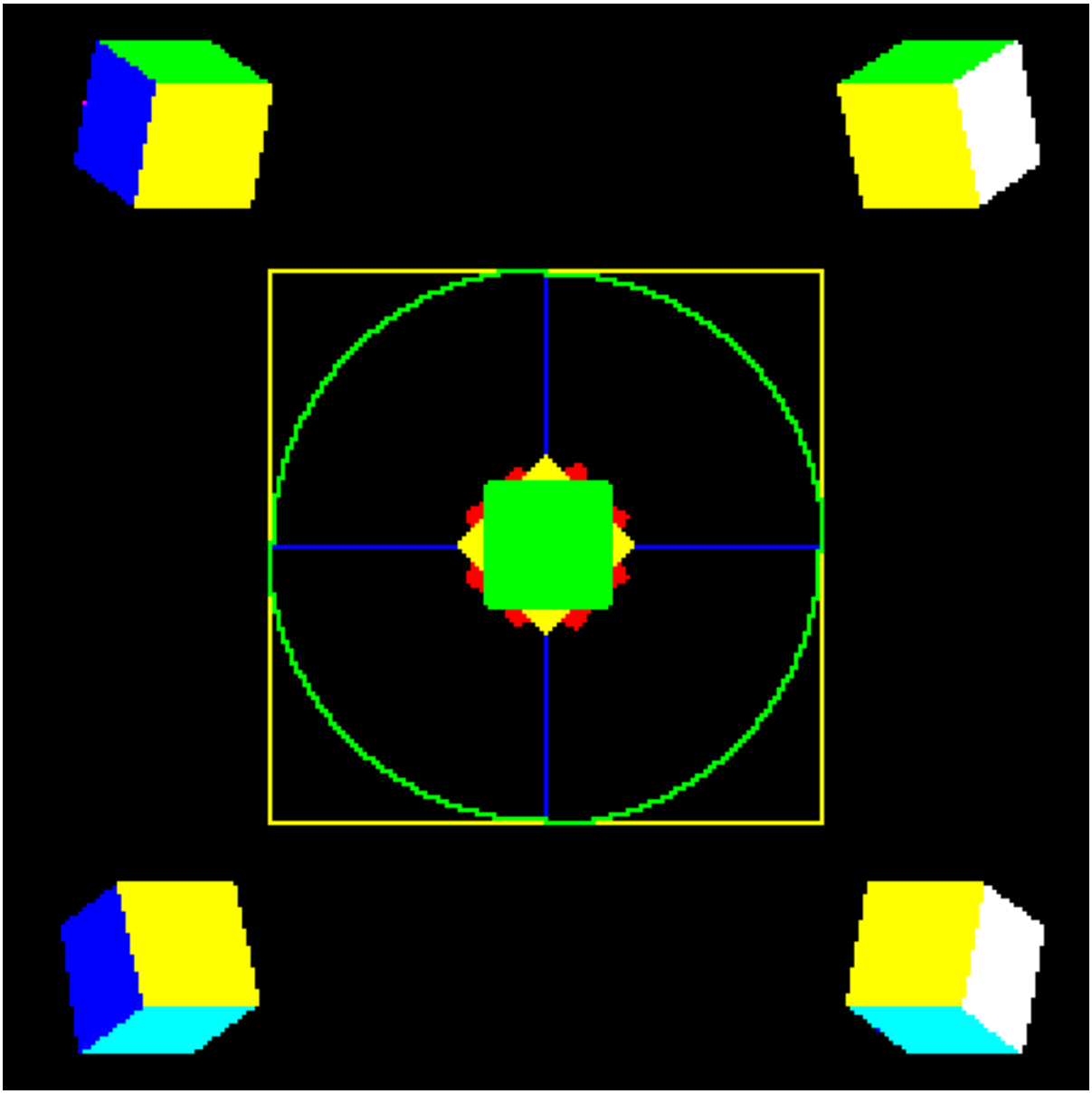
Lucas
SECRET
16/10/20

Rapport POO C++



Algorithmes

Calcul du Cercle

Pour afficher un Cercle, j'ai premièrement calculé tous les points composants la courbe, quartier par quartier. On part du point haut du cercle (x0, y0+rayon) et on calcule l'équation du cercle 0 = a*x + b – y, puis on regarde si pour le point actuel l'équation est supérieure ou inférieure à 0. En dessous de la courbe, on avance x de 1 pour atteindre la courbe, et au-dessus de la courbe on descend y pour atteindre la courbe. On ajoute le point dans un vector. On répète l'opération pour les 3 autres quarts de cercle en changeant les valeurs d'incrémentations.
Une autre méthode affichera le Cercle à partir du vector retourné par la précédente méthode.

```
vector<Point> Circle::getPoints()
{
    vector<Point> pointList;
    Point currentPoint(xAxis, yAxis - radius, 0, color);

    //Top Right Quarter
    while (currentPoint.getXaxis() < xAxis + radius || currentPoint.getYaxis() < yAxis)
    {
        pointList.push_back(currentPoint);

        float equation = pow(currentPoint.getXaxis() - xAxis, 2) + pow(currentPoint.getYaxis() - yAxis, 2) - pow(radius, 2);
        if (equation < 0)
            currentPoint.setXaxis(currentPoint.getXaxis() + 1);

        else
            currentPoint.setYaxis(currentPoint.getYaxis() + 1);

    }

    //Bottom Right Quarter
    while (currentPoint.getXaxis() > xAxis || currentPoint.getYaxis() < yAxis + radius)
    {
        pointList.push_back(currentPoint);

        float equation = pow(currentPoint.getXaxis() - xAxis, 2) + pow(currentPoint.getYaxis() - yAxis, 2) - pow(radius, 2);
        if (equation > 0)
            currentPoint.setXaxis(currentPoint.getXaxis() - 1);

        else
            currentPoint.setYaxis(currentPoint.getYaxis() + 1);

    }
```

Calcul des points d'un Segment

Nous aurons besoin de ces informations afin de remplir plus tard une facette, et donc de tester si un point est dans un segment. Pour cela, de même que pour le cercle, nous allons créer un vector qui contiendra tous les points entre un point de départ et un point d'arrivée, suivant l'équation y = a*x + b.

```
//Calculate all points that have been made for the segment between start and end
void Segment::calculateAllPoints()
{
    allPoints.clear();
    Point currentPoint;
    Point destination;
    bool segmentOnZ = false;



    //Change x for z if x and y are equals
    if (start.getXaxis() == end.getXaxis()
        && start.getYaxis() == end.getYaxis())
    {
        currentPoint.setXaxis(start.getZaxis());
        currentPoint.setYaxis(start.getYaxis());
        destination.setXaxis(end.getZaxis());
        destination.setYaxis(end.getYaxis());

        segmentOnZ = true;
    }


    //Chose the left point as start, the top point if x is equal for the two points
    if (currentPoint.getXaxis() < destination.getXaxis()
        || (currentPoint.getXaxis() == destination.getXaxis() && currentPoint.getYaxis() < destination.getYaxis()))
    {
        currentPoint = start;
        destination = end;
    }
    else
    {
        Point temp = currentPoint;
        currentPoint = destination;
        destination = temp;
    }
```

Premièrement on regarde si le segment en question n'est pas uniquement sur la profondeur Z (points x et y des points de départ et d'arrivée sont égaux), auquel cas on inverse les Z et les X afin de pouvoir s'adapter pour la 2D, pour ne pas faire un calcul spécialement pour ce type de segment.

Ensuite, on regarde quel point est le plus à gauche de l'autre, car dans la suite pour atteindre le point d'arrivée nous allons seulement incrémenter X, et déplacer Y au-dessus ou en dessous de l'équation du segment.

On trouve ensuite le ratio 'a' et l'ordonnée à l'origine 'b' de l'équation y=a*x+b.

```
currentPoint.setColor(color);

//The 'a' in the y=ax + b
double ratio;

//avoid division by zero
if (currentPoint.getXaxis() == destination.getXaxis())
    ratio = 0;
else
    ratio = (double)(((double)destination.getYaxis() - (double)currentPoint.getYaxis()) / ((double)destination.getXaxis() - (double)currentPoint.getXaxis()));
    // a = yb - ya / xb - xa

//The 'b' in y=ax + b
double b = currentPoint.getYaxis() - ratio * currentPoint.getXaxis();

while ((int)currentPoint.getXaxis() != (int)destination.getXaxis() || (int)currentPoint.getYaxis() != (int)destination.getYaxis())
```

On regarde ensuite le signe du ratio 'a', et le signe de l'équation 0 = a*x + b – y avec le x et le y du point concerné. On décide alors si on doit incrémenter ou décrémenter y, puis on ajoute le point au vector.

Si le segment était uniquement sur Z, on ré-échange alors les coordonnées X et Z du point.

```
while ((int)currentPoint.getXaxis() != (int)destination.getXaxis() || (int)currentPoint.getYaxis() != (int)destination.getYaxis())
{
    //ax + b - y = 0
    double equation = ratio * currentPoint.getXaxis() + b - currentPoint.getYaxis();

    //Descending line
    if (ratio > 0)
    {
        if (equation > 0) //so we're under the line
            currentPoint.setYaxis(currentPoint.getYaxis() + 1);

        else
            currentPoint.setXaxis(currentPoint.getXaxis() + 1);
    }

    //Rising line
    else if (ratio < 0)
    {
        if (equation <= 0) //so we're under the line
            currentPoint.setYaxis(currentPoint.getYaxis() - 1);
        else
            currentPoint.setXaxis(currentPoint.getXaxis() + 1);
    }

    //If ratio == 0, it means x or y are equals for the two points (for straight line)
    else
    {
        if (currentPoint.getXaxis() == destination.getXaxis())
            currentPoint.setYaxis(currentPoint.getYaxis() + 1);

        else
            currentPoint.setXaxis(currentPoint.getXaxis() + 1);
    }

    Point result;

    //re-exchange x and z
    if (segmentOnZ)
        result = Point(currentPoint.getZaxis(), currentPoint.getYaxis(),
            currentPoint.getXaxis());
    else
        result = currentPoint;
```

Remplissage d'une facette

Pour remplir l'intérieur d'une face, il a été utilisé un algorithme récursif, qui part d'un point au centre de la face, qui colorie ce point et qui s'appelle elle-même avec les 4 points voisins du point central (Nord, Est, Sud et Ouest), qui eux-mêmes appelleront cette fonction avec leurs points voisins etc.. Jusqu'à ce que le point courant rencontre un des segments de la Face, auquel cas il sort de la fonction récursive.

```
int SquareFace::displayFaceRecursively(vector<Point>& displayedPoint, Point currentPoint, Ppm& image)
{
    /*If face is only on zAxis, no try to display
    (if the cube has no rotation and the face is not the front one or the back one) */
    if (!faceIsVisible())
        return 0;
    //avoid program exit
    if (!currentPoint.isOutOfBounds(image))
    {
        image.setpixel(currentPoint.getXaxis(), currentPoint.getYaxis(), color);
        displayedPoint.push_back(currentPoint);
    }
    else return 0;

    //Check if the point is on the border
    if (pointIsOnEdge(currentPoint))
        return 0;

    //Check if neighbors are already displayed before to call or not the method
    if (!pointIsAlreadyDisplayed(displayedPoint, currentPoint.getNorthNeighbor()))
        displayFaceRecursively(displayedPoint, currentPoint.getNorthNeighbor(), image);

    if(!pointIsAlreadyDisplayed(displayedPoint, currentPoint.getEastNeighbor()))
        displayFaceRecursively(displayedPoint, currentPoint.getEastNeighbor(), image);

    if (!pointIsAlreadyDisplayed(displayedPoint, currentPoint.getSouthNeighbor()))
        displayFaceRecursively(displayedPoint, currentPoint.getSouthNeighbor(), image);

    if (!pointIsAlreadyDisplayed(displayedPoint, currentPoint.getWestNeighbor()))
        displayFaceRecursively(displayedPoint, currentPoint.getWestNeighbor(), image);

    return 0;
}
```

On test d'abord si la face est visible, c'est-à-dire que l'on peut voir la face. Une face qu'on ne peut pas voir est une face qui est seulement dans la profondeur, autrement dit au moins un de ses segments à un X et un Y égaux pour le point de départ et d'arrivée. Si elle n'est pas visible, rien ne sers de l'afficher. On dessine ensuite le point si celui-ci ne sors pas de l'image.

La 3ème condition fait appel à la fonction getPoints() de Segment présentée plus haut, afin de savoir si un point est sur un segment. Pour cela on récupère tous les points de chaque segment de la face, et on test avec le point courant. S'il est effectivement sur un des bords alors on sort de la récursivité pour ce point.

Le dernier exercice de ce TP consistait à écrire un algorithme qui trie les facettes du cube afin d'afficher celle que l'on est censé voir par-dessus celle que nous sommes censés ne pas voir via la perspective.

Pour faire cela, la classe Cube contient un tableau de Face à afficher. On va donc trier le tableau en fonction de la moyenne sur la profondeur de chaque Face.

```
56  void Cube::sortFaces()
57  {
58      for (int i = 0; i < numberOfFace; i++)
59          cout << faces[i].getDepthAverage() << " ";
60
61      for (int i = 0; i < numberOfFace; i++)
62      {
63          int minZcenter = faces[0].getDepthAverage();
64          int position = 0;
65          for (int j = 0; j < numberOfFace - i; j++)
66          {
67              if (faces[j].getDepthAverage() <= minZcenter)
68              {
69                  //Find minimum depth
70                  minZcenter = faces[j].getDepthAverage();
71                  position = j;
72              }
73          }
74          //And put it in the end of the tab
75          SquareFace tempFace(faces[numberOfFace - i - 1]);
76          faces[numberOfFace - i - 1] = faces[position];
77          faces[position] = tempFace;
78      }
79  }
```

C'est un algorithme de tri classique, qui parcourt le tableau, trouve le minimum moyen Z de chaque face, puis qui, une fois arrivé à la fin du tableau, intervertit le dernier élément du tableau avec la Face ayant le minimum moyen Z. On recommence l'opération avec le même tableau mais de taille-1, car le dernier élément du tableau est forcément le plus petit.

On pourra de cette façon afficher l'ensemble des Faces via une boucle for, sans s'occuper de placer à la main les Faces qui s'afficheront par-dessus les autres.