Rasterization et gestions des faces cachées

TD n°3

Dans ce TP, vous allez repartir du projet précédent et vous allez y ajouter deux fonctions principales pour la suite : la *rasterization* et un algorithme pour la gestion des faces cachées.

Dans ce projet, il vous faudra exclusivement compléter les parties du code source où le commentaire suivant est marqué

```
// TODO => TP03 //
```

1 L'algorithme de *rasterization*

L'algorithme se décompose en trois parties.

1. La fonction `draw_quad()` sera la fonction coordinatrice de l'ensemble de ce travail ; elle a pour fonction de déterminer les contours du polygone.
2. La fonction `raster_buffer_insert()` a pour fonction de mémoriser les x minimum et maximum pour chaque y dans un buffer spécifiquement dédié.
3. La fonction `draw_horizontal_line()` est la fonction la plus simple qui a simplement pour mission de dessiner une ligne horizontale sur un y déterminé entre les deux bornes x donné.

L'idée de la *rasterization* est de déterminer les contours du polygone grâce à l'algorithme de dessin de ligne de Bresenham afin d'enregistrer, pour chaque y rencontré, les bornes minimum et maximum sur l'axe x. Nous prendrons comme hypothèse importante que tous les polygones rencontrés sont convexes (le cas des polygones concaves est plus complexe à traiter).

Pour mettre en place cet algorithme, on utilisera un tampon `raster_buffer` de la taille de la hauteur maximale de l'image (pour couvrir tous les y possibles) et de largeur 2 (pour la borne minimum et la borne maximum sur l'axe x). La fonction `raster_buffer_insert()` aura pour mission de déterminer pour chaque nouveau point du contour du polygone s'il doit être sauvegarder comme borne minimum ou maximum pour la hauteur y considérée. Pour initialiser ce tampon, on pourra utiliser la valeur `MININT` qui est une valeur neutre pour identifier que les cases ne sont pas remplies.

Une fois le tampon `raster_buffer` rempli, il suffira d'envoyer les bornes à la fonction `draw_horizontal_line()` pour remplir ligne par ligne le polygone.

Exercice 1 *Vous commencerez par remplir la fonction suivante*

```
draw_horizontal_line(int y, int x1, int x2, vec3 c);
```

qui prend en argument la hauteur `y` *de la ligne dessinée puis les bornes minimum* `x1` *et maximum* `x2`. `c` *est la couleur de la ligne.*

Vous pouvez très facilement tester cette fonction.

Exercice 2 *Complétez la fonction de remplissage du tampon permettant la rasterization*

```
void raster_buffer_insert(int x, int raster_buffer[2]);
```

où `x` *est la valeur courante insérée dans le tampon et* `raster_buffer` *est le tableau de sortie pour les bornes minimum et maximum identifiées.*

Exercice 3 *Complétez la fonction permettant de déterminer le contour du polygone puis dessinant les lignes horizontales*

```
void draw_quad(vec2 p[4], vec3 c);
```

où `p` *est la liste des points du polygone et* `c` *est la couleur du polygone.*

Vous pouvez très facilement tester cette fonction.

2 Algorithme des faces cachées

L'algorithme des faces cachées consiste à identifier rapidement quelles faces seront visibles depuis la caméra et lesquelles seront cachées. Nous prendrons comme hypothèse forte que l'objet observé est convexe et fermé ; de cette façon, nous allons pouvoir identifier les faces invisibles grâce à leur normale et au point de vue de la caméra.

L'opération permettant de déterminer si la face est visible ou non doit permettre de mettre à jour l'attribut `visible` de chaque face de l'objet dessiné. Cet attribut est décrit dans la classe `Object`.

Afin de tester cet algorithme, nous allons dessiner un cube dont les faces sont de couleurs différentes. De la même façon que la sphère lors du précédent TP, ce cube va tourner, montrant alternativement les différentes faces. Décommentez les lignes correspondant au TP03 qui se trouve dans la fonction `display()` du fichier `main.cpp`.

Exercice 4 *Complétez le cas `DRAW_FILL` de la fonction `draw()` dans le fichier `object.cpp`. Ce mode de dessin doit permettre de dessiner l'ensemble des polygones en les remplissant (rasterization) avec leur couleur.*

Exercice 5 *Complétez la fonction permettant de déterminer la visibilité des faces afin de remplir le champ `visible` des instances de la classe `Object`*

```
void backface_culling(mat4 trans_matrix);
```

où `trans_matrix` est la matrice de transformation de la scène.