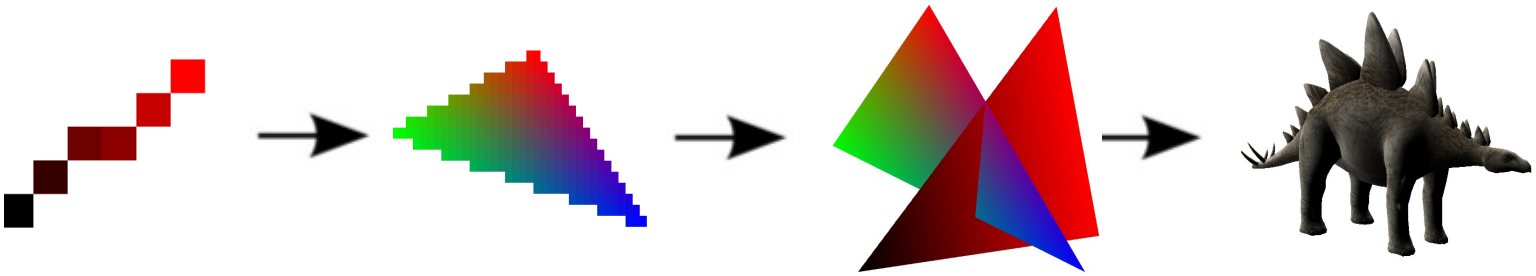
EMERY Guillaume
ZIZA Simon

05/09/2016

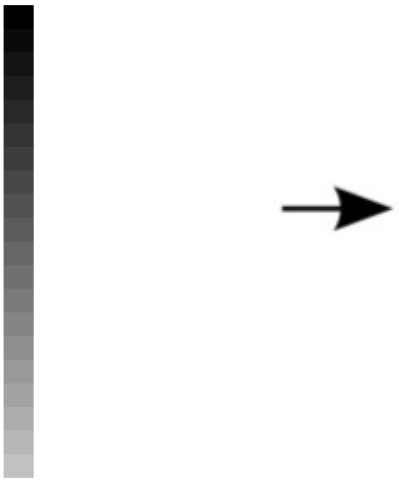
Rendu Projectif



I Segments et Contours de Triangle

a) Ligne verticale

Nous traçons dans un premier temps un simple segment vertical à x constant et observons le fichier



Ligne verticale

```
px   R    G    B
480 255  255  255
481 255  255  255
482 255  255  255
483 255  255  255
484 255  255  255
485 255  255  255
486 255  255  255
487 255  255  255
488 255  255  255
489 91  91  91
490 255  255  255
491 255  255  255
492 255  255  255
493 255  255  255
494 255  255  255
```

Code en ASCII de l'image

En regardant le code ASCII, on conclut que le format *ppm* enregistre les pixels de l'image ligne par ligne. C'est pourquoi les pixels qui se suivent en colonne ne se suivent pas dans le fichier en ASCII.

	mon_image.jpg	mon_image.png	mon_image.ppm
Nom :	mon_image.jpg	mon_image.png	mon_image.ppm
Type :	image JPEG (image/jpeg)	image PNG (image/png)	image PPM (image/x-portable-pixmap)
Taille :	650 octets	158 octets	12,0 ko (11 985 octets)

Comparaisons de 3 formats d'image, *jpg*, *png* et *ppm*.

On remarque en effet que les formats *jpg* et encore plus *png*, créent des fichiers de bien plus petite taille (~18 fois pour le *jpg* et ~76 fois pour le *png*).

La classe Image dérive de la classe template ImageBasic où seront implémentées les méthodes attributs.

b) Segments quelconques couleur uniforme

On teste l'algorithme de Bresenham entre [5,5] et [12,9]. On est donc dans le premier octant. Nous obtenons un résultat cohérent, qui suit les règle de l'algorithme vu en cours :

```
(5 ; 5) (6 ; 6) (7 ; 6) (8 ; 7) (9 ; 7) (10 ; 8) (11 ; 8) (12 ; 9)
```

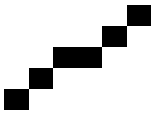
On a bien des pixels de [5,5] à [12,9] avec un déplacement de un au maximum sur chaque coordonnée. L'affichage de ce segment va permettre de vérifier qu'il n'y a pas de pixel affiché en trop :

Si on essaie ensuite de tracer sur un autre octant, les fonctions de symétries n'étant pas implémentées, on passe dans un *assert()* qui vérifie que la taille du segment est positive et le programme stoppe son exécution. Deux autres assertions permettent de vérifier l'intégrité du segment à tracer.

```
simon.ziza@tpc12:~/Documents/5A/Rendu_projectif/tp_projectif/projet/bin$ ./pgm
**********************
run ./pgm with 0 parameters ...

(5 ; 5) (6 ; 6) (7 ; 6) (8 ; 7) (9 ; 7) (10 ; 8) (11 ; 8) (12 ; 9)
*********************************
*********************************
Exception found
ASSERT FAILED: [line.size()>0] with message: Incorrect size of line in Bresenham ( line.size()=0 )
In file [/fs03/share/users/simon.ziza/home/Documents/5A/Rendu_projectif/tp_projectif/projet/src/discrete/bresenham.cpp]
In function [bresenham]
At line 68


Saving backtrace in file backtrace_log.txt
*********************************
*********************************
```

Assertion failed pour la taille du segment

On modifie alors le code pour effectuer les symétries nécessaires au tracé dans chaque octant. Pour vérifier le code, nous traçons un segment dans les autres quadrant.
Par exemple le quadrant 8 :



b) Segments quelconques / interpolation de couleurs

Nous cherchons désormais à tracer une ligne entre un point p0 et p1 telle que la couleur varie linéairement entre c0 et c1.

Nous affichons les paramètres d'interpolation qui varient entre 0 et 1 avec
$\forall k \in [0:N], \quad \alpha_k + \alpha_{N-k} = 1$.

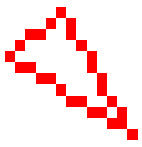
```
simon.ziza@tpc12:~/Documents/5A/Rendu_projectif/tp_projectif/projet/bin$ ./pgm
**********************
run ./pgm with 0 parameters ...

(5 ; 5) (6 ; 6) (7 ; 6) (8 ; 7) (9 ; 7) (10 ; 8) (11 ; 8) (12 ; 9)
0 0.175412 0.27735 0.447214 0.5547 0.723241 0.83205 1
Exit Main
```

Avec un seul point le paramètre peut valoir n'importe quelle valeur entre 0 et 1 mais il est fixé à 0 dans notre programme.

On affiche un segment quelconque avec une couleur interpolée du noir au rouge :

c) Contours d'un triangle quelconque de couleur uniforme

Pour tracer le contour d'un triangle, il suffit de tracer 3 segments :



II Triangles plein et interpolation de couleurs

a) Algorithme scanline

```
simon.ziza@tpc24:~/Documents/5A/Rendu_projectif/tp_projectif/projet/bin$ ./pgm
**********************
run ./pgm with 0 parameters ...

x=3 , [(6 ; 3) , 0 0 1] - [(6 ; 3) , 0 0 1]
x=4 , [(6 ; 4) , 0 0.0658472 0.934153] - [(7 ; 4) , 0.111111 0 0.888889]
x=5 , [(5 ; 5) , 0 0.150472 0.849528] - [(8 ; 5) , 0.222222 0 0.777778]
x=6 , [(5 ; 6) , 0 0.216924 0.783076] - [(9 ; 6) , 0.333333 0 0.666667]
x=7 , [(5 ; 7) , 0 0.282957 0.717043] - [(10 ; 7) , 0.444444 0 0.555556]
x=8 , [(5 ; 8) , 0 0.348442 0.651558] - [(11 ; 8) , 0.555556 0 0.444444]
x=9 , [(4 ; 9) , 0 0.433648 0.566352] - [(12 ; 9) , 0.666667 0 0.333333]
x=10 , [(4 ; 10) , 0 0.5 0.5] - [(13 ; 10) , 0.777778 0 0.222222]
x=11 , [(4 ; 11) , 0 0.565628 0.434372] - [(14 ; 11) , 0.888889 0 0.111111]
x=12 , [(3 ; 12) , 0 0.649798 0.350202] - [(15 ; 12) , 1 0 0]
x=13 , [(3 ; 13) , 0 0.716824 0.283176] - [(13 ; 13) , 0.83946 0.16054 0]
x=14 , [(3 ; 14) , 0 0.782814 0.217186] - [(11 ; 14) , 0.678919 0.321081 0]
x=15 , [(3 ; 15) , 0 0.846426 0.153574] - [(8 ; 15) , 0.453219 0.546781 0]
x=16 , [(2 ; 16) , 0 0.93132 0.0686803] - [(5 ; 16) , 0.226735 0.773265 0]
x=17 , [(2 ; 17) , 0 1 0] - [(3 ; 17) , 0.0666544 0.933346 0]

Exit Main
```

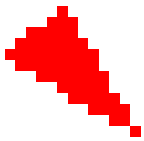
Affichage de la structure scanline

L'algorithme parcourt le triangle ligne par ligne de haut en bas en cherchant les coordonnées des points aux extrémités de chaque segment composant le triangle. Grâce à la fonction template *triangle_scanline_factory()*, on pourra entrer en paramètres n'importe quel type de variable à la place des couleurs (par exemple la profondeur). On pourra interpoler ces valeurs.

On accède aux extrémités de chaque segment grâce aux deux variables *left* et *right*. Ces variables sont composées d'un *ivec2* (coordonnées du point) et d'un paramètre de type *template*, qui correspond à une caractéristique du point (couleur, profondeur, coordonnées texture, etc.).

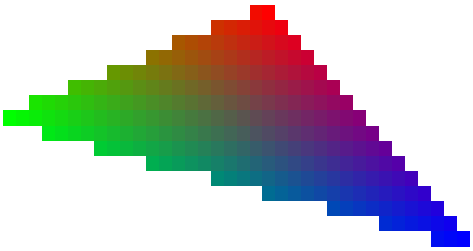
b) Triangle plein uniforme

On commence par dessiner un triangle plein de couleur uniforme.
(Un seul paramètre de couleur, qui sera utilisé pour les trois points).



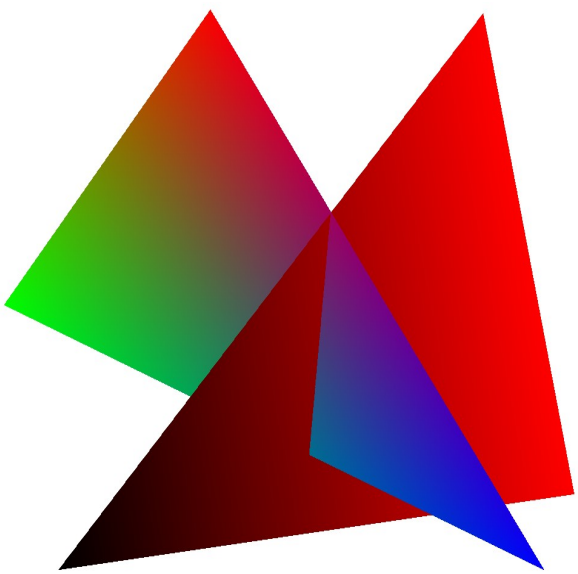
c) Triangle plein couleurs interpolées

On peut ensuite utiliser la fonction *triangle_scanline_factory()* pour tracer un triangle où les couleurs sont interpolées.

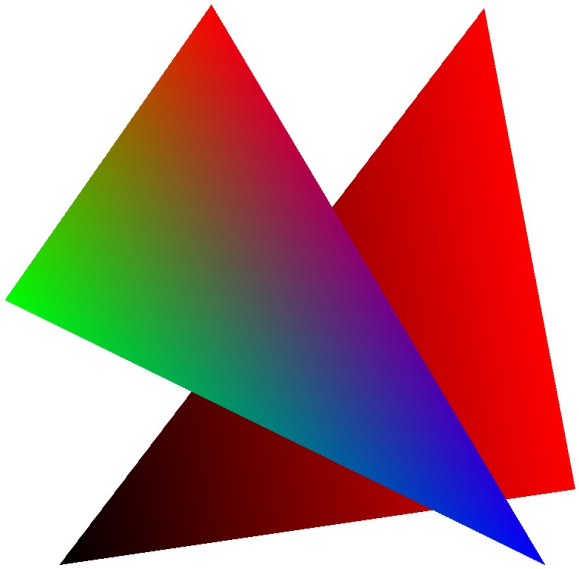


III Gestion de la profondeur

Il faut créer un buffer (appelé le zbuffer) qui va enregistrer la profondeur de chaque pixel affiché à l'écran. A chaque nouveau pixel que l'on veut dessiner sur l'écran, on va vérifier que sa profondeur est comprise entre 1 et -1 et inférieure à la valeur du zbuffer associé à ce pixel. Si cette condition est remplie, alors on écrit le pixel dans l'image, et on met à jour le zbuffer avec la profondeur de ce pixel.
Ainsi, il faut initialiser toutes les profondeur du zbuffer à 1, valeur qui représente le fond de la scène.



Triangles croisés

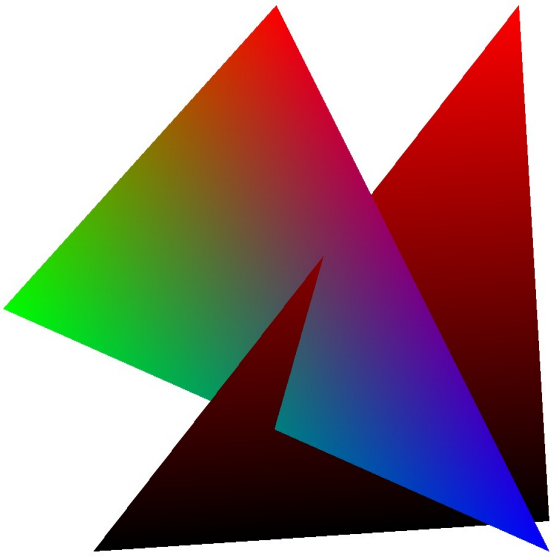


Triangles superposés

On peut maintenant tracer plusieurs triangles de profondeurs variables.

IV Projection

En utilisant 3 matrices identités pour les matrices *model*, *view* et *projection*, on retrouve le résultat ci dessus.



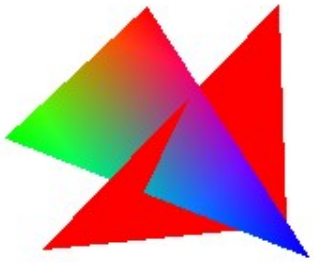
Remarque : nous avons changé les couleurs du deuxième triangle, au lieu d'avoir un angle noir et deux angles rouges, nous avons mis un angle rouge et deux noirs.

Maintenant, nous rentrons des paramètres de *projection*. Il faut également adapter la matrice *view* pour rester dans le champ de vision de la caméra.

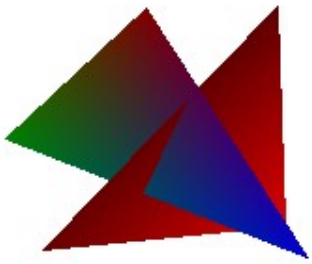
Les triangles semblent plus éloignés de la caméra.

V Illumination

On utilise les paramètres Ka (facteur ambiant), Kd (facteur diffus), Ks (facteur spéculaire) pour rendre l'illumination à l'écran.



Ka = 1

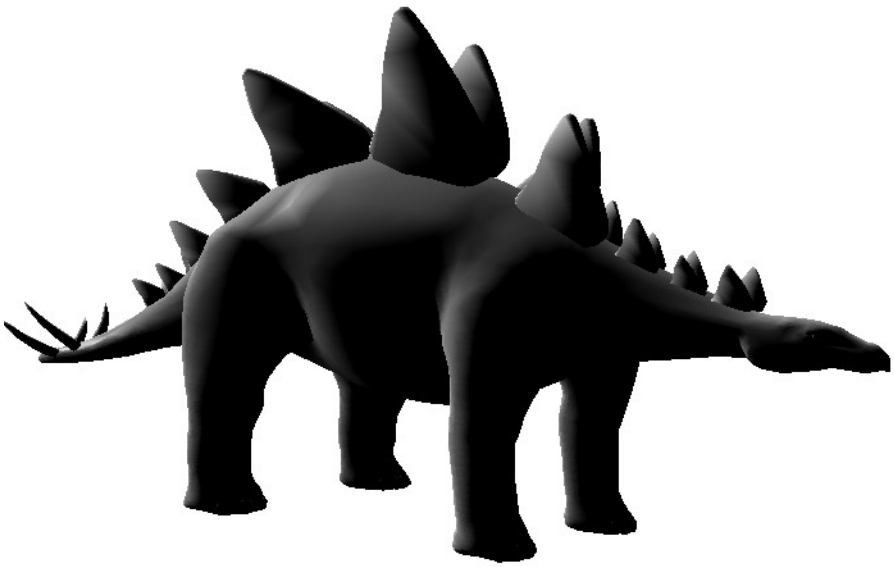


Ka = 0.2

Le facteur ambiant permet de modifier l'éclairage sur toute l'image.

VI Maillage et Texture

Après avoir compléter le code, on peut afficher un maillage à l'écran.



Maillage sans texture

Une couleur blanche est appliquée sur l'ensemble du maillage et les normales sont calculées automatiquement.

Ensuite, en ajoutant en paramètres la texture et les coordonnées de texture aux fonctions codées tout au long du TP, on peut appliquer une image en texture du maillage.