PAN
Julien
17802782

RAPPORT DE PROJET : CASTLE et PITTEWAY

Sommaire :

1. Implémentation de l'algorithme de référence tel que **Bresenham 65.**

```
void affiche(int x, int y){
   printf("%d \t %d\n", x, y);
}
void bresenham(int u, int v){
   int x, y, delta, incH, incD;
   incH = v << 1;
   delta = incH - u;
   incD = delta - u;
   for(x = 0, y = 0; x <= u; x++){
       //affiche(x, y);
       if(delta > 0){
           y++;
           delta += incD;
       }
       else
           delta += incH;
   }
}
```

Après l'implémentation de l'algorithme de Bresenham (l'algorithme de référence), j'ai pu implémenter l'algorithme de Castle et Pitteway qui est sur le site du cours.

Implémentation de l'algorithme de Castle et Pitteway.

```
char * castle(int dx, int dy){
   int i = 1;
   dx -= dy;
   char* s = malloc(sizeof(char));
   s = "0";
   char* t = malloc(sizeof(char));
   t = "1";
   char * res;
   while(dx != dy){
       if(dx > dy){
           dx -= dy;
           res = (char *) malloc(1 + strlen(s) + strlen(t));
           strcpy(res, s);
           strcat(res, t);
           t = res;
       }
       else{
           dy -= dx;
           res = (char *) malloc(1 + strlen(s) + strlen(t));

           strcpy(res, s);
           strcat(res, t);
           s = res;
       }
   }
   char * resFinal = (char *) malloc(1 + strlen(s) + strlen(t));
   strcpy(resFinal, s);
   strcat(resFinal, t);
     char * resFinalDxFois = (char *) malloc(dx * (strlen(s)+
strlen(t)));
   strcpy(resFinalDxFois, resFinal);
   while(i < dx){
       strcat(resFinalDxFois, resFinal);
       i++;
   }
   //printf("%s \n", resFinalDxFois);
```

```
    return resFinalDxFois;
}
    //printf("%s \n", resFinalDxFois);
    return resFinalDxFois;
}
```

Pour cette algorithme, il faut concaténer et dupliquer grâce) **strcat()** et **strcpy()**, le nombre de fois donné pour avoir la droite qui sera formé à partir de 0 et de 1 à la suite.
Un exemple : castle(20,11) = 01010101010101010101

2. Tests de temps et de mémoire

J'ai pu tester et comparer en temps les deux algorithmes :

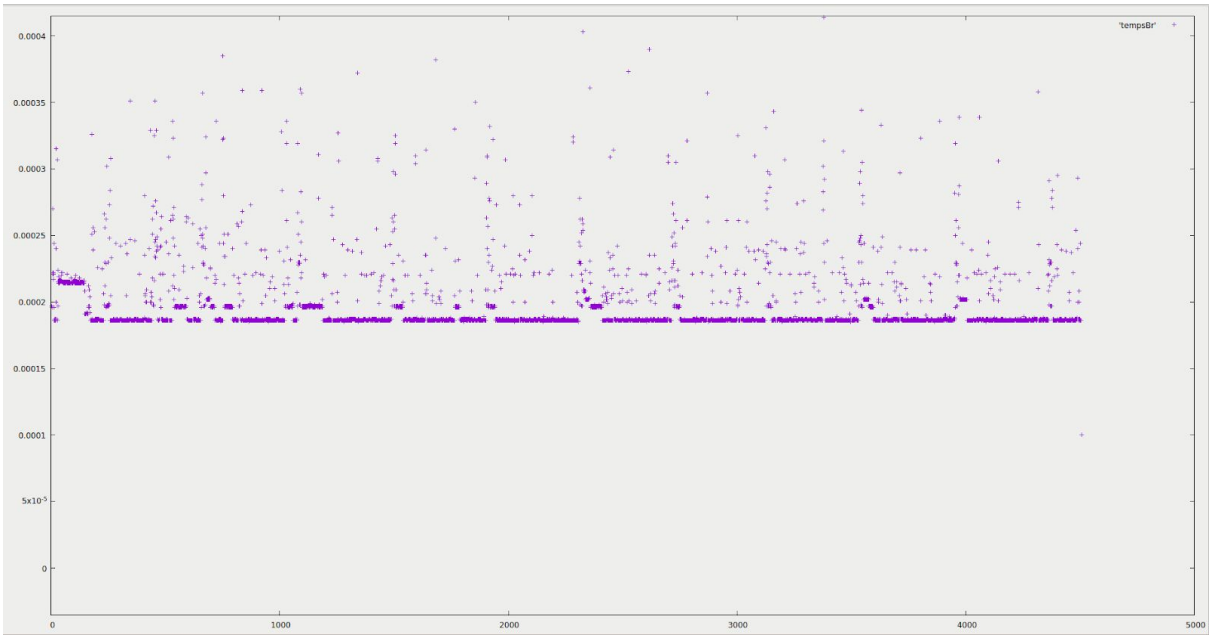
```
int main(void){
    clock_t start_t, end_t, start2_t, end2_t;
    float time_t, time2_t;
    start_t = clock();
    bresenham(1000000, 900000);
    end_t = clock();
    time_t = (float)(end_t - start_t)/CLOCKS_PER_SEC;
     printf("temps d'execution de l'algorithme de Bresenham: %f
ms\n", time_t);

    start2_t = clock();
    char * d = castle(1000000, 900000);
    end2_t = clock();
    time2_t = (float)(end2_t - start2_t)/CLOCKS_PER_SEC;
      printf("temps d'execution de l'algorithme de Castle et
Pitteway: %f ms\n\n", time2_t);
    return 0;
}
```

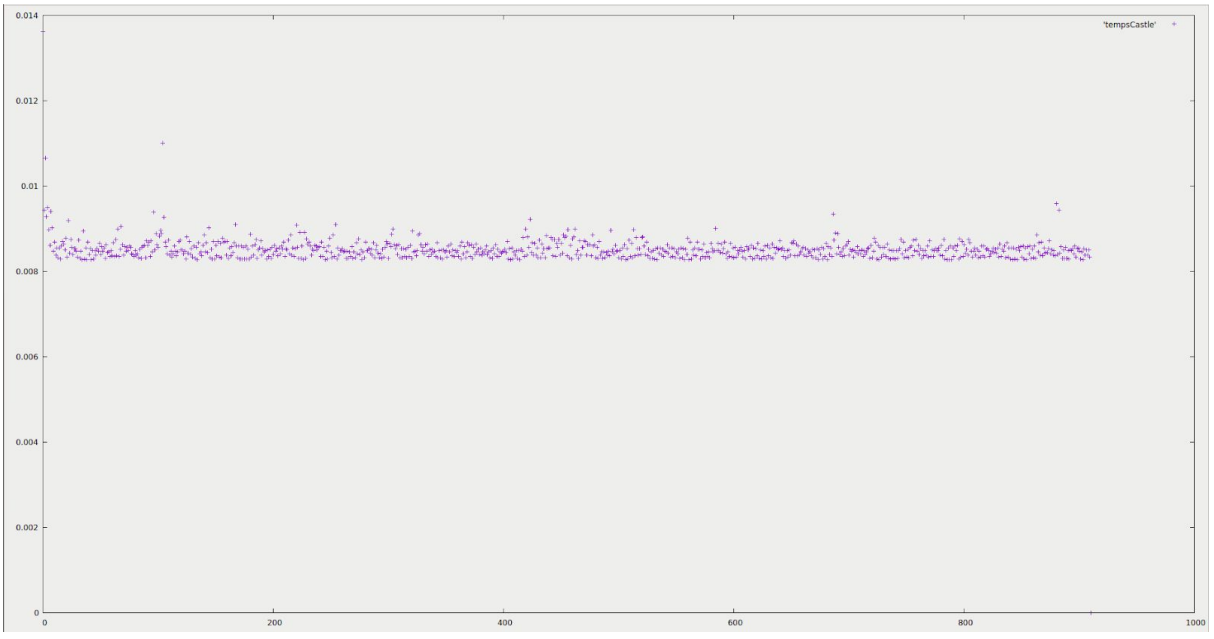
Ce qui nous affiche :
temps d'exécution de l'algorithme de Bresenham: **0.004476 ms**
temps d'exécution de l'algorithme de Castle et Pitteway: **1.112892 ms**

Les graphes :

Bresenham :



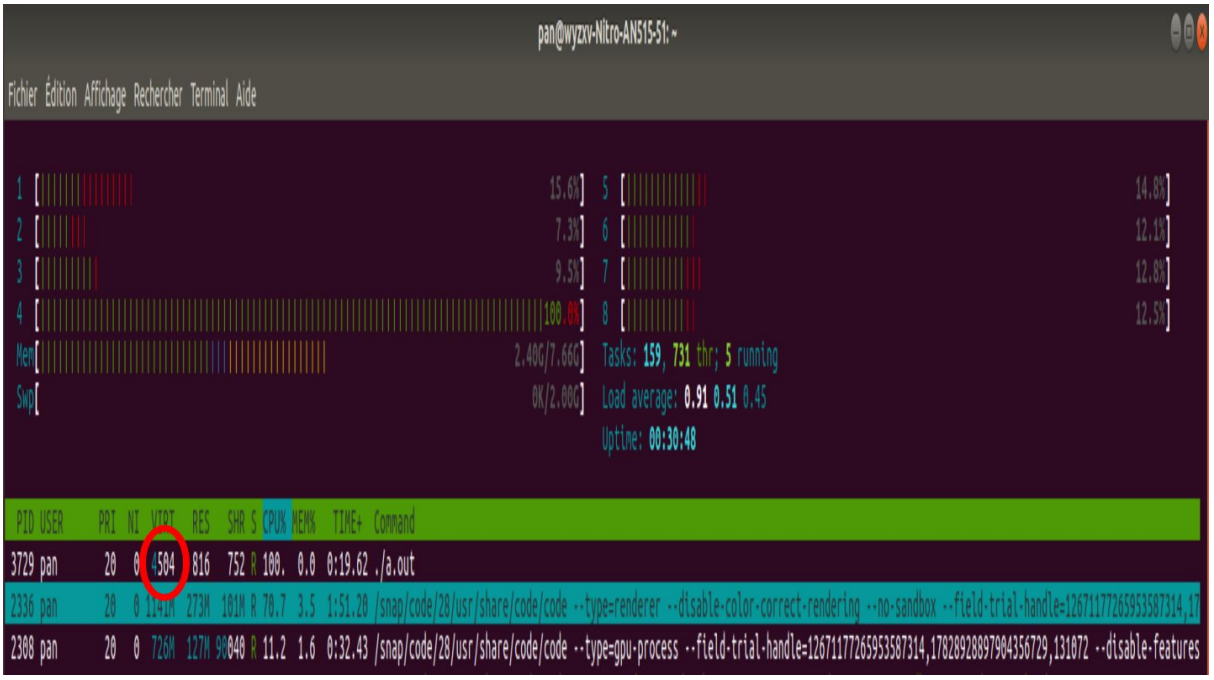
Castle et Pitteway :



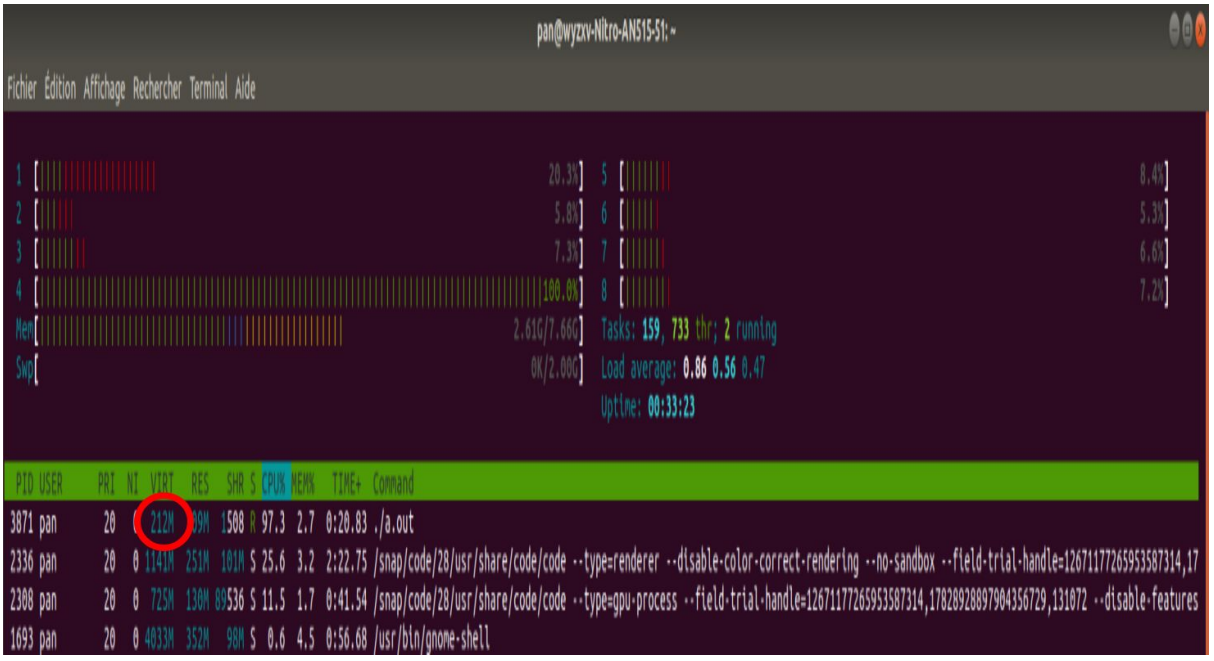
On voit que le temps d'exécution de Castle et Pitteway est plus lente mais reste constante et que Bresenham est plus rapide.

Utilisation en mémoire :

Pour l'algorithme de Bresenham, on remarque que l'utilisation de la mémoire est constante peu importe la taille des valeurs passées en paramètres. Ici la mémoire utilisé est de 4504 kb, ce qui est peu.



On peut voir que l'algorithme de Caste et Pitteway utilise beaucoup plus de mémoire quand il s'agit de grandes valeurs. Cette valeur augmente en fonction du temps. Ici la mémoire utilisé est de 212 Mkb...



Côté amélioration, je n’ai pas réussi à faire cette partie par mes propres moyens.

Pour conclure, la comparaison de ces deux algorithmes permettent de voir que Bresenham est plus rapide pour de grandes valeurs, et que Castle et Pitteway est plus rapide pour des petites valeurs. Quant à la mémoire utilisée, Castle et Pitteway utilise beaucoup plus avec le temps pour de grandes valeurs alors que Bresenham reste constant peu importe la taille des valeurs.