

# Synthèse d'images - Compte rendu de TP5

## Introduction

Dans la fenêtre nous visualisons un rectangle qui prends toute la place de la fenêtre et qui produit un dégradé de couleurs.

Les positions des sommets sont transmises par VAO et VBO.

Les couleurs que nous voyons sont des couleurs donnée à partir des positions des pixels.

## Fractale de Mandelbrot

Calculer le carré d'un complexe

```
vec2 square(in vec2 a){  
    vec2 res;  
    res.x = a.x*a.x - a.y*a.y;  
    res.y = 2*a.x*a.y;  
    return res;  
}
```

Calcul de couleur à partir d'un entier

```
vec4 colormap(in float n){  
    float r,g,b;  
    r = 1-n;  
    g = 1-n;  
    b = cos(n);  
    return vec4(r,g,b,1.0);  
}
```

Calcul de mandelbrot

```
float mandelbrot(in vec2 c, in int N){  
    float S = 1000.;  
    int div = 10;  
    vec2 z = vec2(0.0);  
  
    for(int i = 0; i < N; ++i){  
        z = square(z) + c;  
        if(length(z) > S){  
            return float(i)/float(N-1);  
        }  
    }  
    return 1.0;  
}
```

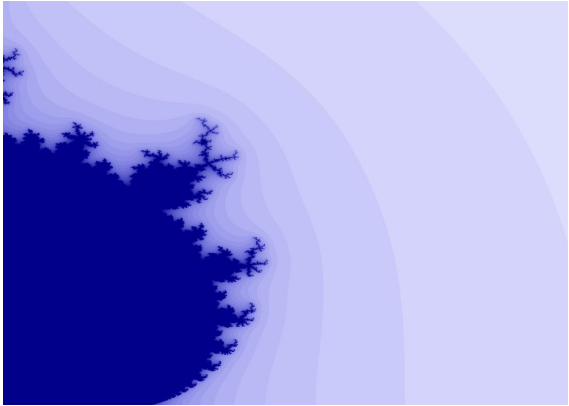
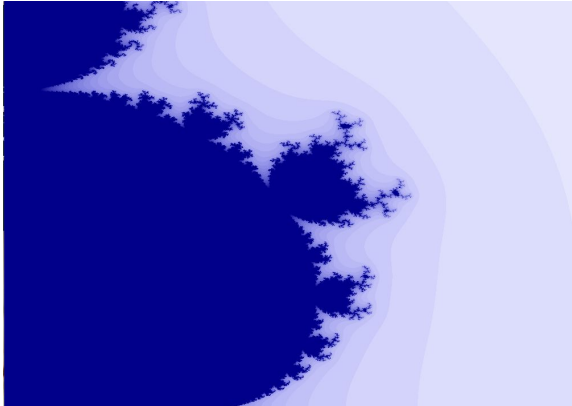
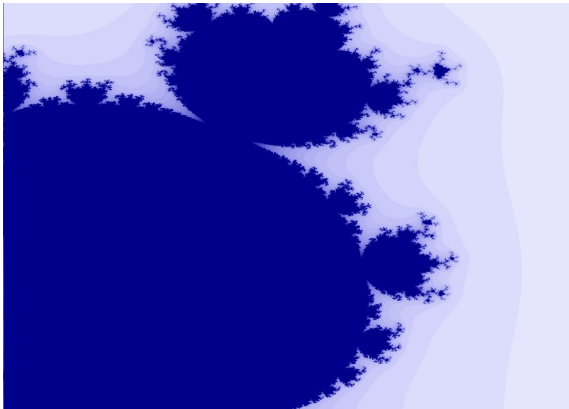
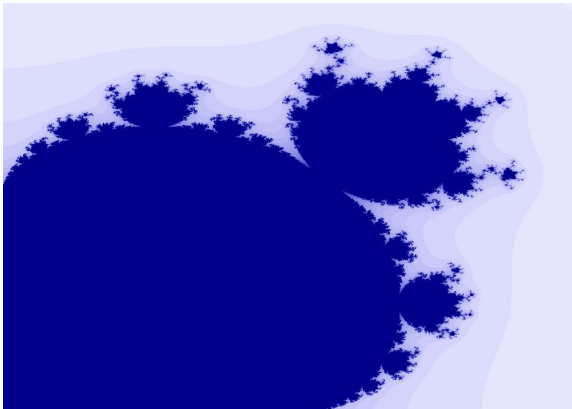
## Ensemble de Mandelbrot d'ordre K

Pour pouvoir calculer  $Z$  à la puissance  $k$  nous avons écrit le code suivant :

```
vec2 mult(in vec2 a, in vec2 b){
    vec2 res;
    res.x = a.x*b.x - a.y*b.y;
    res.y = a.x*b.y + a.y*b.x;
    return res;
}

// Calcule la puissance p d'un nombre complexe
vec2 pow(in vec2 c, in int p) {
    vec2 res = c;
    for( int i = 1; i < p; i++){
        res = mult(res,c);
    }
    return res;
}
```

Nos résultats ont été les suivants :

Ordre 2	Ordre 3
	
Ordre 4	Ordre 5
	

# Paramètres uniformes

## Animation

Pour passer le temps on ajoute dans main.cpp

```
GLuint timeID = glGetUniformLocation(programID, "time");
```

Avant la boucle, et

```
cur_time = glfwGetTime() - init_time;  
glUniform1f(timeID, cur_time);
```

Dans la boucle.

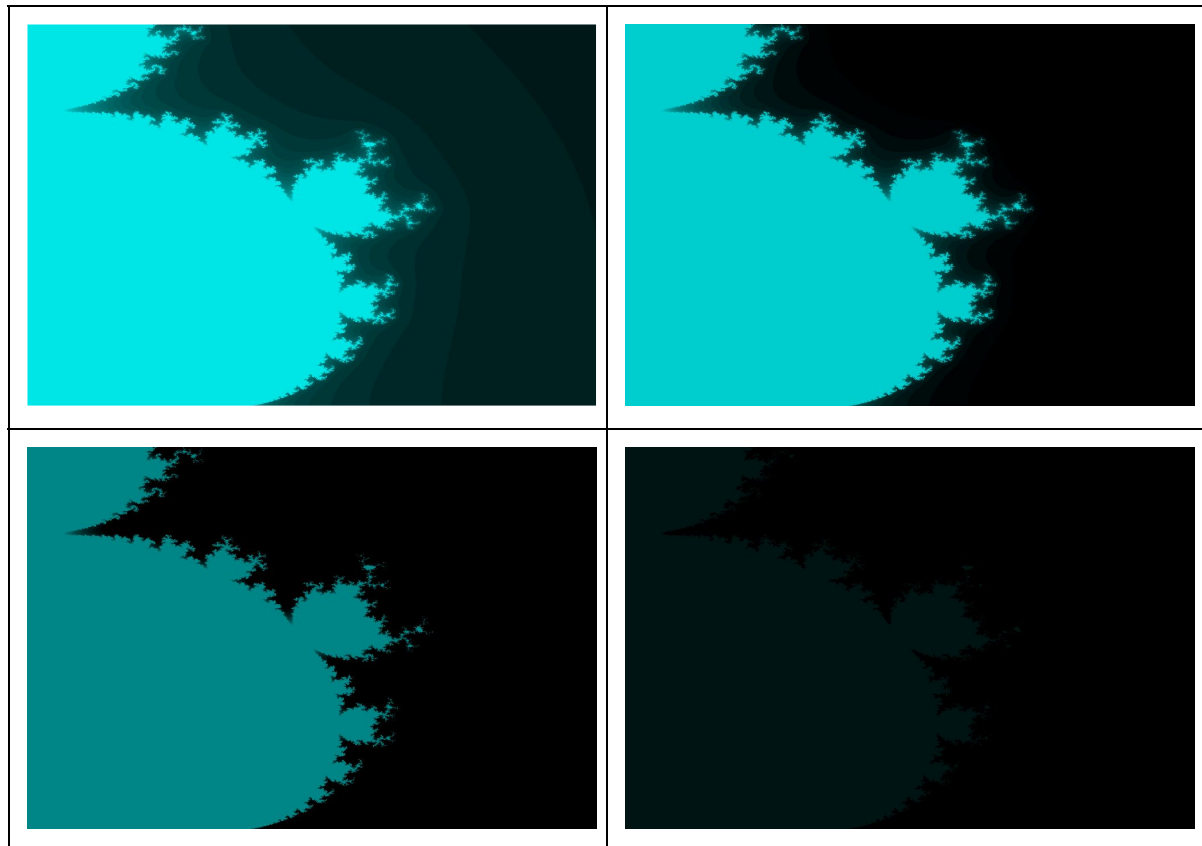
Cela va permettre d'exporter la variable cur\_time aux shaders.

Ensuite dans le fragment shader, pour accéder à la variable on ajoute

```
uniform float time;
```

Nous pouvons ensuite utiliser comme bon nous semble cette variable afin de modifier le comportement de la fractale. (position en commentaire ou couleur)

On obtient donc une évolution de la couleur de la fractale en fonction du temps :



## Contrôle de la caméra

Afin de pouvoir observer la fractale, il est possible de simuler le fait de se déplacer dans le plan. Nous commençons par définir 4 float permettant de délimiter la zone d'affichage à l'écran dans main.cpp :

```
p0x = -1.0;  
p0y = 1.0;  
p1x = 1.0;  
p1y = -1.0;
```

On peut ensuite faire varier leurs valeurs ou niveau de la boucle lors de l'appuis sur des touches. Nous avons utilisé une fonction controle\_camera() appelée avant chaque dessin de la fractale :

```
void controle_camera(){  
    if (glfwGetKey( GLFW_KEY_UP ) == GLFW_PRESS) {  
        p0y = p0y-0.02;  
        p1y = p1y-0.02;  
    }  
    if (glfwGetKey( GLFW_KEY_DOWN ) == GLFW_PRESS) {  
        p0y = p0y+0.02;  
        p1y = p1y+0.02;  
    }  
    [...]  
    if (glfwGetKey( GLFW_KEY_S ) == GLFW_PRESS) {  
        p0x = p0x + 0.02;  
        p0y = p0y - 0.02;  
        p1x = p1x - 0.02;  
        p1y = p1y + 0.02;  
    }  
    if (glfwGetKey( GLFW_KEY_Z ) == GLFW_PRESS) {  
        p0x = p0x - 0.02;  
        p0y = p0y + 0.02;  
        p1x = p1x + 0.02;  
        p1y = p1y - 0.02;  
    }  
}
```

De la même façon que précédemment, on exporte les variable aux shaders :

```
glUniform1f(p0xID,p0x);  
glUniform1f(p0yID,p0y);  
glUniform1f(p1xID,p1x);  
glUniform1f(p1yID,p1y);
```

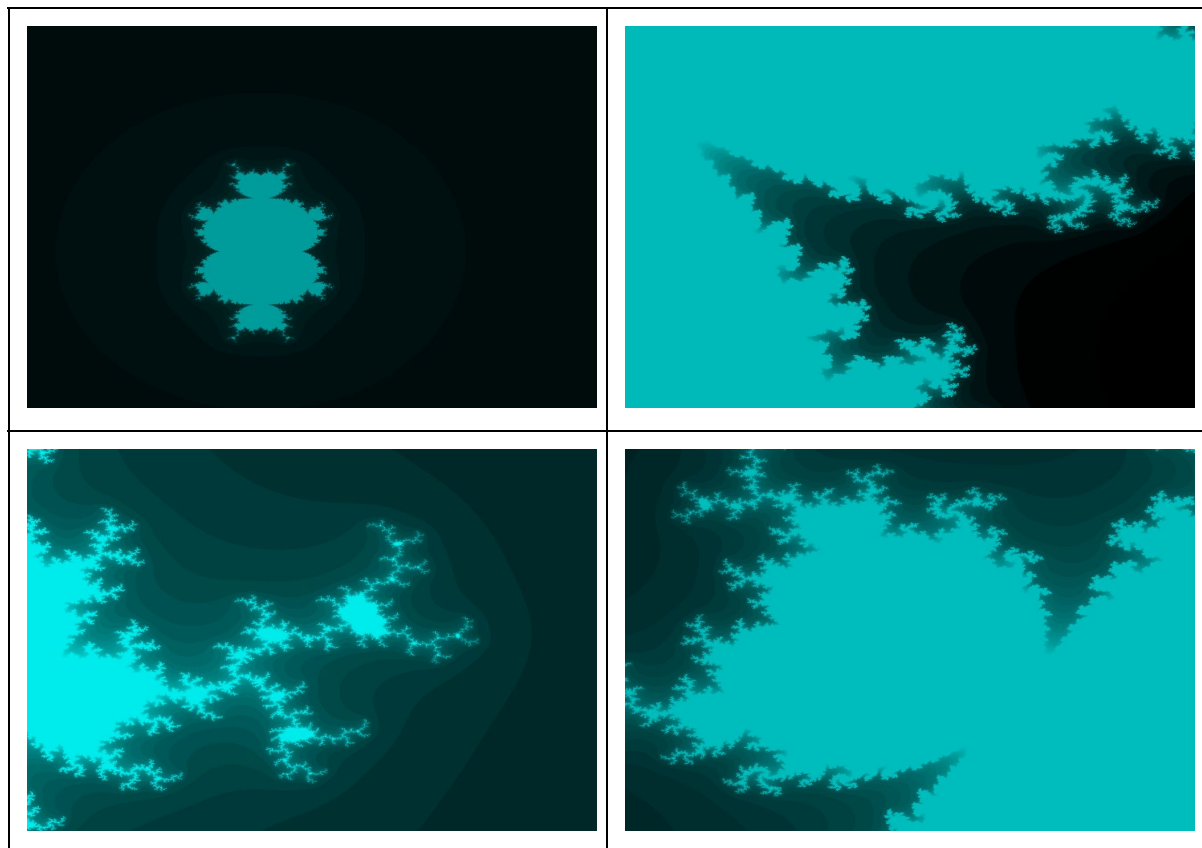
Vertex.glsl transforme les quatre float en un vec2 indiquant le point de départ de la fractale, ce qui simule le déplacement ainsi que le zoom.

```
coords = vec2((in_position.x-p0x)/(p1x-p0x),(in_position.y-p1y)/(p0y-p1y));
```

Ce nouvel élément peut ensuite être directement utilisé dans la fonction mandelbrot(in vec2 **c**, in int N) utilisée précédemment pour la génération de la fractale :

```
[...]  
for(int i = 0; i < N; ++i){  
    z = pow(z,3) + c;  
    if(length(z) > S){  
        return float(i)/float(N-1);  
    }  
}
```

Le résultat permet de se déplacer dans le plan et de zoomer :



À noter que notre zoom ne fonctionne pas totalement car à moins d'être centré sur le point initial de la fractal, la fenêtre a tendance à se déplacer dans la direction de l'origine de la fractale au zoom, et à s'en éloigner au dézoom. Nous n'avons pas réussi à résoudre cette erreur qui vient certainement d'une erreur de calcul du centre actuel de la fenêtre...