

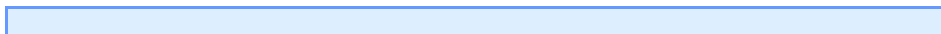
Chapitre XIV : Scrolling avec SDL



par [Loka](#)

Date de publication : 29/10/2007

Dernière mise à jour : 29/10/2007



XIV - Scrolling

XIV-1 - Scrolling de la caméra

XIV-2 - Scrolling du background

Sources et pdf

Remerciements

XIV - Scrolling

Jusqu'à maintenant, nous avons travaillé avec des environnements en 640 par 480 (de la taille de notre fenêtre SDL).

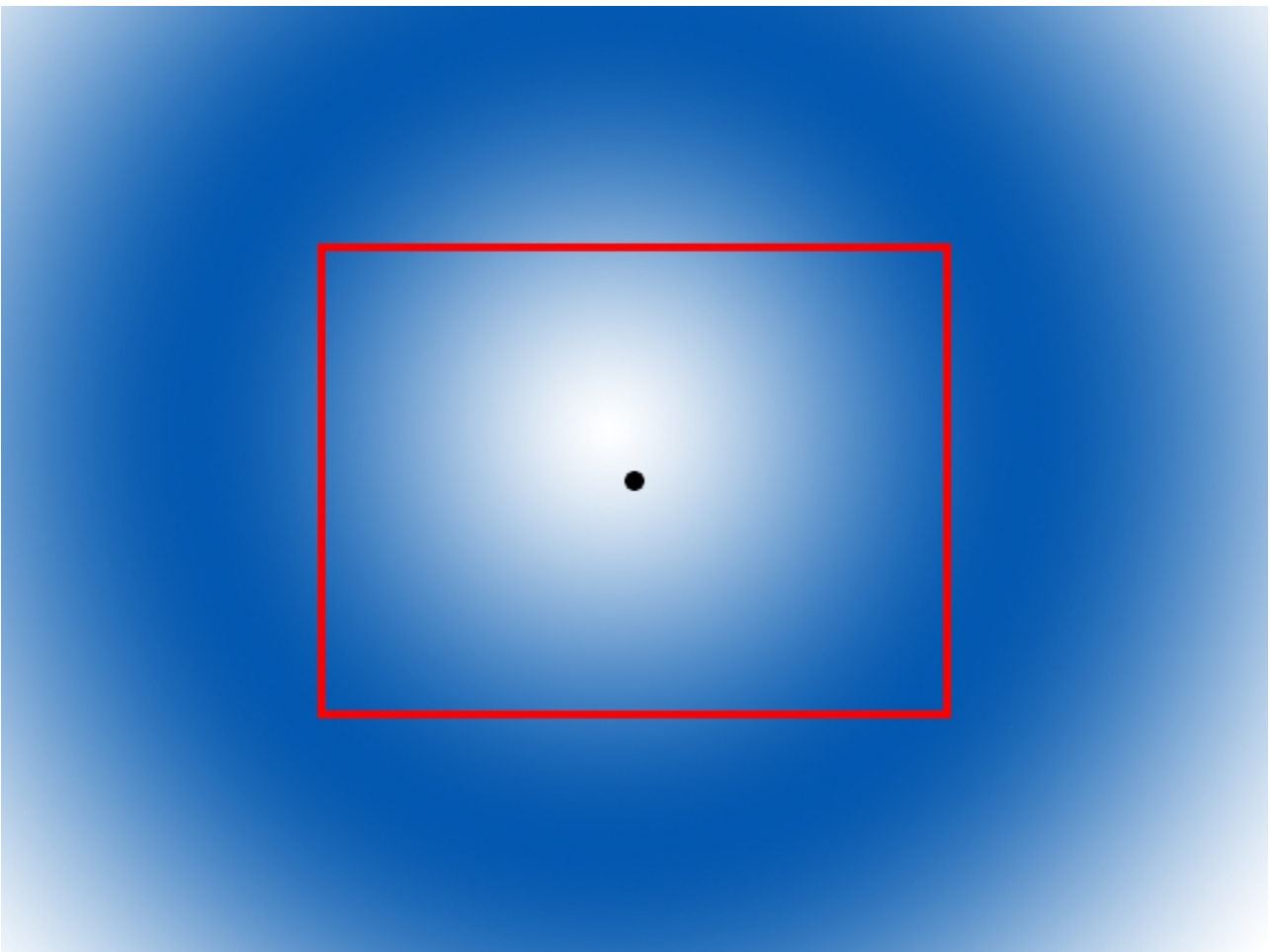
Ce tutoriel va vous apprendre à vous déplacer à travers des environnements de n'importe quelle taille en utilisant le principe du scrolling (défilement).

XIV-1 - Scrolling de la caméra

Quand on fait un jeu qui requiert du scrolling, la seule différence est qu'il nous faut ce qu'on appelle une caméra.

Vu qu'on ne peut pas afficher le niveau en entier à l'écran qui est trop petit, il nous suffit de prendre la partie du niveau qu'on veut afficher (généralement celle contenant notre élément actif).

Par exemple sur l'image suivante, la partie qu'on va afficher est entourée en rouge :



Il faut penser aussi que la caméra doit suivre notre point sur l'image si celui se met à bouger.

Dans cette partie du tutoriel, nous allons créer ce programme.

Dans la déclaration des variables, on va voir apparaître quelques nouveautés.

variables

```
//Les attributs de l'écran (640 * 480)
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_BPP = 32;

//Le nombre de frame par seconde
const int FRAMES_PER_SECOND = 30;

//Les dimensions du point (l'image qu'on va déplacer)
const int POINT_WIDTH = 20;
const int POINT_HEIGHT = 20;

//Vitesse de déplacement du point (en pixel par seconde)
const int POINT_VITESSE = 200;

//Les dimensions du niveau
const int LEVEL_WIDTH = 1280;
const int LEVEL_HEIGHT = 960;

//Les surfaces
SDL_Surface *point = NULL;
SDL_Surface *background = NULL;
SDL_Surface *screen = NULL;

//La structure d'événements que nous allons utiliser
SDL_Event event;

//La camera
SDL_Rect camera = { 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT };
```

Comme vous le voyez, on a définie les dimensions de notre niveau en plus de celles de la fenêtre SDL, celle-ci étant plus petite que le niveau.

On voit aussi apparaître la caméra qui est tout simplement un `SDL_Rect` car de forme rectangulaire.

On initialise cette caméra à la taille de notre fenêtre SDL car ce que l'on va afficher sera de la taille de notre fenêtre SDL.

Encore une fois, nous allons devoir revoir notre classe `Point` afin d'y ajouter la caméra :

classe Point

```
class Point
{
private:
//Les coordonnées x et y du point
int x, y;

//La vitesse du point
int xVel, yVel;

public:
//Initialisation des variables
Point();

//Recupere la touche pressee et ajuste la vitesse du point
void handle_input();

//Bouge le point
void move();

//Montre le point sur l'ecran
void show();

//Met la camera sur le point
void set_camera();
```

```

classe Point
};

```

C'est la même chose qu'avant, sauf qu'on ajoute une fonction **set_camera()** pour centrer la caméra sur notre point.

Autre chose que l'on va retoucher légèrement est la fonction **move()** de notre Point.

En effet, on ne va plus tester les collisions avec les bords de notre fenêtre SDL et donc de l'écran mais avec les bords du niveau.

```

move()

```

```

void Point::move()
{
    //Bouge le point à gauche ou à droite
    x += xVel;

    //Si le point se rapproche trop des limites(gauche ou droite) de l'ecran
    if( ( x < 0 ) || ( x + POINT_WIDTH > LEVEL_WIDTH ) )
    {
        //On revient
        x -= xVel;
    }

    //Bouge le point en haut ou en bas
    y += yVel;

    //Si le point se rapproche trop des limites(haute ou basse) de l'ecran
    if( ( y < 0 ) || ( y + POINT_HEIGHT > LEVEL_HEIGHT ) )
    {
        //On revient
        y -= yVel;
    }
}

```

Comme vous pouvez le voir, la différence par rapport à avant est minime.

Nous ne gardons donc plus notre point dans l'écran mais dans notre niveau.

Nous allons maintenant regarder la fonction **set_camera()**.

```

set_camera()

```

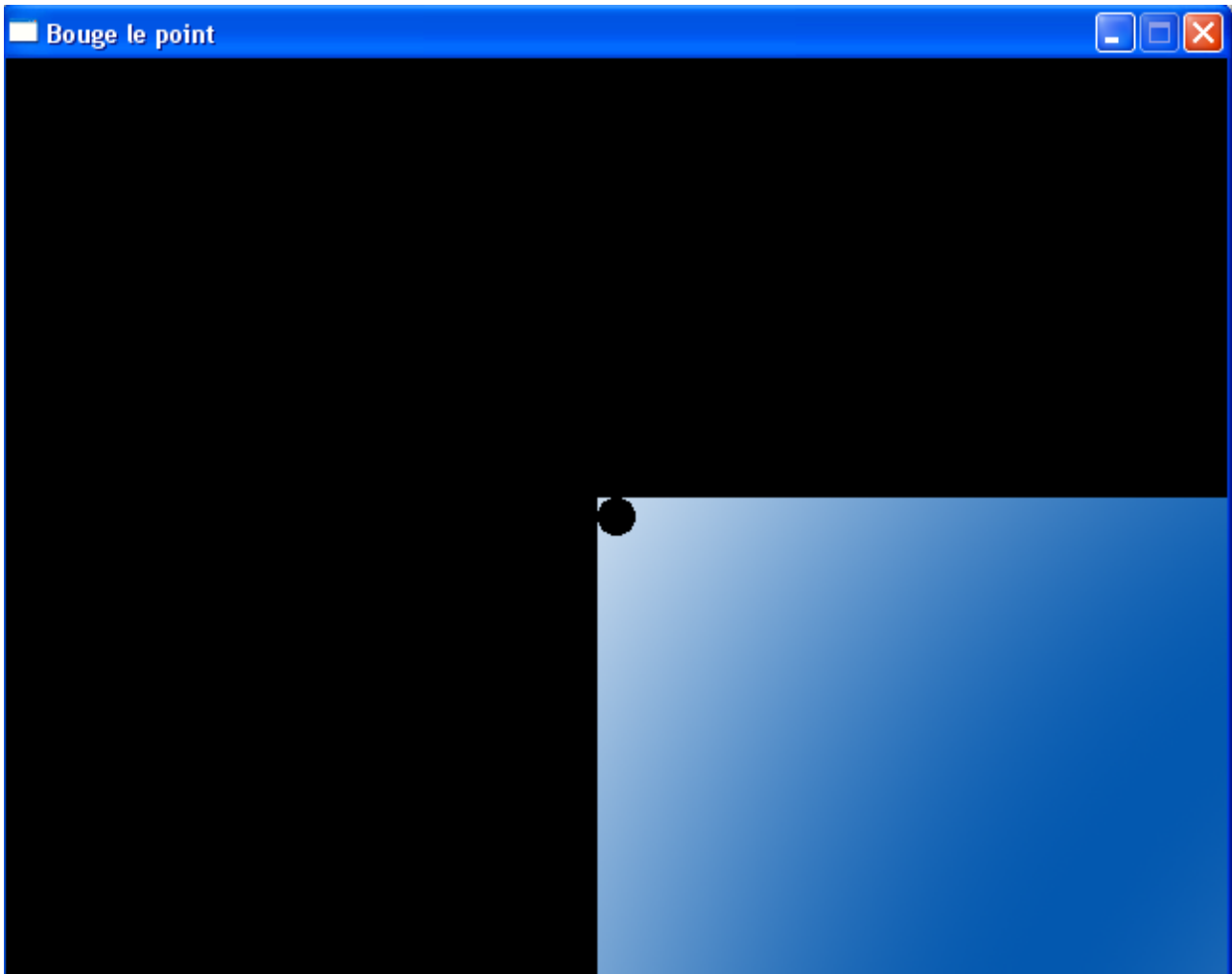
```

void Point::set_camera()
{
    //Centre la camera sur le point
    camera.x = ( x + POINT_WIDTH / 2 ) - SCREEN_WIDTH / 2;
    camera.y = ( y + POINT_HEIGHT / 2 ) - SCREEN_HEIGHT / 2;
}

```

Quand nous mettons en place la caméra, dans un premier temps nous la centrons autour du point (le point étant le centre de la caméra).

Cependant si nous ne faisons que ça, quand le point va se rapprocher des bords du niveau, on va voir apparaître du vide comme c'est le cas sur l'image ci-dessous :



Pour éviter cela, on va arrêter le mouvement de la caméra lorsque le point arrive vers les extrémités du niveau.

Cela aura pour conséquence de ne plus voir de vide et de voir le point continuer à bouger alors que la caméra s'est arrêtée.

set_camera() : suite

```
//Garde la camera sur les bords
if( camera.x < 0 )
{
    camera.x = 0;
}
if( camera.y < 0 )
{
    camera.y = 0;
}
if( camera.x > LEVEL_WIDTH - camera.w )
{
    camera.x = LEVEL_WIDTH - camera.w;
}
if( camera.y > LEVEL_HEIGHT - camera.h )
{
    camera.y = LEVEL_HEIGHT - camera.h;
}
}
```

Si une partie de la caméra va en dehors du niveau, nous la remettons tout simplement dans le niveau, c'est le même principe de collision avec un objet statique vu dans le [chapitre XI](#).

La fonction pour afficher le point va aussi changer un peu.

En effet, maintenant on va l'afficher sur l'écran relativement à la caméra.

```
show()
void Point::show()
{
    //Affiche le point
    apply_surface( x - camera.x, y - camera.y, point, screen );
}
```

Dans notre boucle principale du main, voici comment ça se déroule :

```
boucle principale
//Tant que l'utilisateur n'a pas quitter
while( quit == false )
{
    //On démarre le timer fps
    fps.start();

    //Tant qu'il y a un événement
    while( SDL_PollEvent( &event ) )
    {
        //On recupere l'evenement pour le point
        monPoint.handle_input();

        //Si l'utilisateur a cliqué sur le X de la fenêtre
        if( event.type == SDL_QUIT )
        {
            //On quitte the programme
            quit = true;
        }
    }

    //Bouge le point
    monPoint.move();

    //Met la camera
    monPoint.set_camera();

    //Affiche le fond
    apply_surface( 0, 0, background, screen, &camera );

    //Affiche le point sur l'écran
    monPoint.show();
}
```

On bouge le point, on met la caméra en place, on affiche la partie du fond dans la caméra puis on affiche le point.

Rien de bien sorcier.

Vous pouvez retrouver les sources du programme de cette partie du tutoriel ici :

[Télécharger les sources du chapitre XIV-1 \(854 ko\)](#)

XIV-2 - Scrolling du background

Il peut arriver qu'on ait besoin d'un niveau de taille infinie, dans ce cas il serait impossible de passer le niveau comme précédemment.

Une solution est de faire défiler un fond en boucle.

Il peut aussi arriver que vous vouliez faire un jeu avec un niveau prédéfini particulièrement long et que ce niveau avance tout seul, comme le célèbre vieux jeu [R-Type](#).

Cette partie du tutoriel a pour but de créer un programme capable de faire un défilement du niveau.

Le défilement horizontal d'un niveau ressemble à quelque chose comme ça :



Il est bien sûr possible de faire un défilement vertical avec le même principe.

En réalité, le fond est blitté de multiples fois comme si une caméra avançait dessus.

Quand on arrive au bout du fond, on le remet au début, ce qui nous donne une impression d'infinie.

Pour débiter, il nous faut donc mettre le fond au départ :

```
coordonnees de depart du fond
//Les coordonnées du fond
int bgX = 0, bgY = 0;
```

Vu que le fond bouge, nous devons garder en mémoire ses coordonnées.

Dans la boucle principale du main, on va bouger ce fond :

boucle principale : déplacement du fond

```
//Tant que l'utilisateur n'a pas quitter
while( quit == false )
{
    //On démarre le timer fps
    fps.start();

    //Tant qu'il y a un événement
    while( SDL_PollEvent( &event ) )
    {
        //Si l'utilisateur a cliqué sur le X de la fenêtre
        if( event.type == SDL_QUIT )
        {
            //On quitte the programme
            quit = true;
        }
    }

    //Défilement du fond
    bgX -= 2;

    //Si le fond est allé trop loin
    if( bgX <= -background->w )
    {
        //On le remet à zero
        bgX = 0;
    }
}
```

Après la récupération des événements, nous déplaçons le fond vers la gauche.

Ensuite nous vérifions si celui-ci est allé trop loin et nous le remettons au début le cas échéant.

Il nous reste à l'afficher :

boucle principale : affichage

```
//Affichage du fond
apply_surface( bgX, bgY, background, screen );
apply_surface( bgX + background->w, bgY, background, screen );

//Affichage du point
apply_surface( 310, 230, point, screen );
```

Nous affichons le fond sur l'écran placé avec ses coordonnées.

Pour mieux voir le rendu de ce défilement, nous ajoutons un point statique (je vous laisse le soin de le rendre dynamique).

Les sources de ce programme sont téléchargeable ici :

[Télécharger les sources du chapitre XIV-2 \(306 ko\)](#)

Sources et pdf

[Télécharger les sources du chapitre XIV-1 \(854 ko\)](#)

[Télécharger les sources du chapitre XIV-2 \(306 ko\)](#)

[Version pdf \(310 ko - 11 pages\)](#) 

Remerciements

Je remercie [Aspic](#) pour sa correction orthographique.