Chapitre XVII : Utilisation d'un Font Bitmap avec SDL

par Loka (autres articles)

Date de publication : 20/05/2008

Dernière mise à jour : 20/05/2008

XVII - Utilisation d'un Font Bitmap....3
XVII-1 - Création d'un font bitmap....3
XVII-2 - Utilisation d'un font bitmap avec SDL....5
Remerciements....11

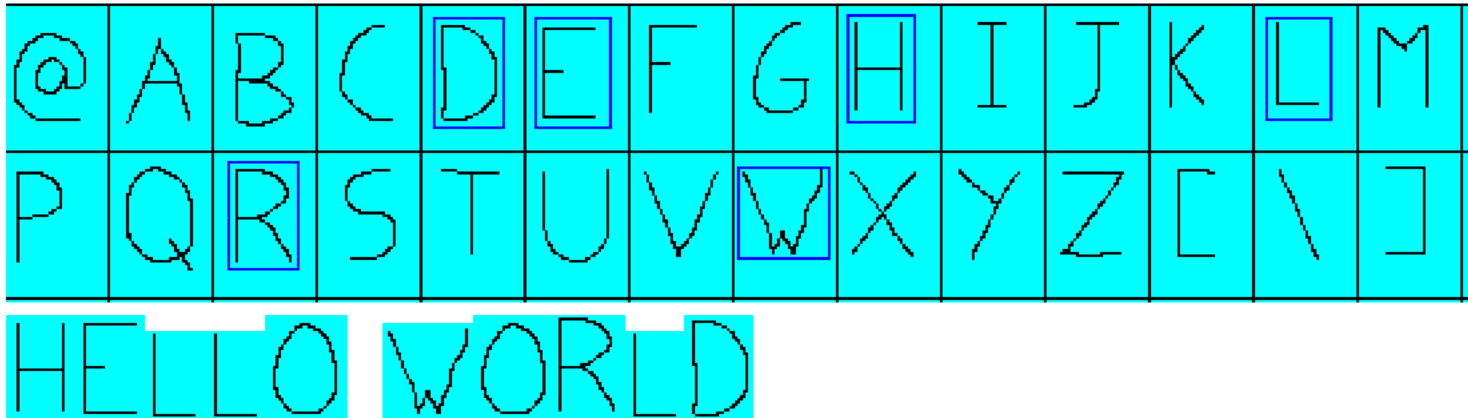
Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

XVII - Utilisation d'un Font Bitmap

Un font bitmap consiste en une série de pixels représentant une image de chaque caractères.
Les fonts bitmap sont utilisées lorsque les bibliothèques de font existantes (comme SDL_ttf) ne sont simplement pas assez flexible.
Ce tutoriel va vous apprendre comment créer un font bitmap et comment s'en servir sous SDL à travers un programme exemple.

XVII-1 - Création d'un font bitmap

Un font bitmap est simplement une feuille de sprites.
Par exemple, si vous souhaitez faire un "HELLO WORLD", il suffit de piocher les lettres qu'on a besoin sur la feuille de sprite pour les afficher comme suit :



Il existe de multiples façons de créer un font bitmap et je vais en citer seulement 2 :
- Utiliser un logiciel spécialisé dans la création de font bitmap (gratuit ou non). J'ai découvert récemment un créateur de font bitmap en ligne et gratuit : **Bitfontmaker**. L'inconvénient avec ce dernier, c'est qu'il restreint la taille de chaque font (caractère) à 10 x 10 pixels.
- Créer le font bitmap à la main en utilisant un logiciel de dessin. C'est cette dernière possibilité que j'ai utilisé pour réaliser la font bitmap pour ce tutoriel.
La première chose à penser lorsqu'on créé un font bitmap est la norme qu'on va choisir. Je vous conseil la norme **ASCII** dans tous les cas.
Si vous souhaitez avoir plus de caractères, il vous faudra regarder du coté des normes ISO comme par exemple **ISO-8859-1** plus connu sous le nom Latin-1 qui est utilisée dans les pays occidentaux.
Si vous souhaitez des caractères plus exotiques, vous pouvez chercher dans les normes **ISO-8859**. Par exemple, pour l'arabe c'est la norme ISO-8859-6.
Pour ma part, je me suis limité aux 95 caractères de la norme ASCII. Cependant, j'ai fais une feuille de sprite pouvant contenir autant de caractères que dans les normes ISO-8859, vous pourrez donc à loisir changer / compléter mon font bitmap comme bon vous semble, celle-ci sera fournie avec le programme exemple.
Pour construire le font bitmap manuellement, je vous conseil, dans un premier temps, de définir la taille de vos caractères afin de connaître la taille de vos cellules qui contiendront vos caractères.
Dans mon cas, chaque cellule fait une taille de 39 x 55 pixels, ce qui est assez large. Comme j'ai fait une feuille assez grande pour contenir autant de caractères que dans les normes ISO-8859, j'ai 256 cellules et donc une feuille d'une taille de 624 x 880 (16 cellules par ligne et 16 cellules par colonnes).
Il vous reste plus qu'à faire une jolie grille pour que vos cellules apparaissent clairement et dessiner vos caractères à l'intérieur de chaque cellule à l'endroit correspondant à la norme choisie.
Voila le résultat de ce que ça peut donner quand c'est fait rapidement à la souris :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

XVII-2 - Utilisation d'un font bitmap avec SDL

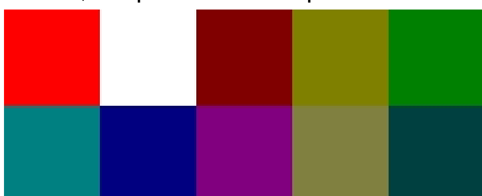
Maintenant que vous savez comment créer un font bitmap ainsi que son principe d'utilisation, il est temps de passer à son utilisation avec SDL.
On va commencer par créer une classe BitmapFont :

classe BitmapFont
```
//Notre classe BitmapFont
class BitmapFont
{
    private:
    //La surface du font
    SDL_Surface *bitmap;

    //Les caractères individuels dans le font
    SDL_Rect chars[ 256 ];

    public:
    //Le constructeur par defaut
    BitmapFont();

    //Le constructeur avec un argument, genere le font quand l'objet est construit
    BitmapFont( SDL_Surface *surface );

    //Genere le font
    void build_font( SDL_Surface *surface );

    //Affiche le texte
    void show_text( int x, int y, std::string text, SDL_Surface *surface );
};
```

La première chose que nous avons dans cette classe est la surface du font bitmap. Ensuite nous avons un tableau de SDL_Rect pour découper les lettres depuis le font bitmap. Nous en avons mit 256 pour pouvoir gérer les normes vues plus haut.
Ensuite nous avons nos constructeurs, une fonction *build_font()* afin d'initialiser le font et pour finir une fonction *show_text()* afin d'afficher le texte à l'écran.
Pour les besoins du programme, nous allons avoir besoin d'une fonction qui nous permet de récupérer un pixel individuel depuis une surface :

get_pixel32()
```
Uint32 get_pixel32( int x, int y, SDL_Surface *surface )
{
    //Convertie les pixels en 32 bit
    Uint32 *pixels = (Uint32 *)surface->pixels;

    //Recupere le pixel demande
    return pixels[ ( y * surface->w ) + x ];
}
```

La première chose que nous faisons dans cette fonction est de convertir le pointeur vers le pixel depuis un type void vers un type entier non signé 32 bits (Uint32) afin de pouvoir y accéder proprement par la suite.
Ensuite, nous récupérons le pixel demandé en argument. Vous vous demandez sûrement pourquoi nous n'avons tout simplement pas récupéré le pixel en retournant juste *pixels[x][y]*.
En fait, les pixels ne sont pas stockés de cette façon :



Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Ils sont stockés ainsi :

Dans un tableau d'une seule dimension.
Donc, pour récupérer le pixel qu'on souhaite depuis le tableau, on multiplie la coordonnées y par la taille et on ajoute la coordonnées x.
Ces fonctions ne marchent qu'avec des surfaces 32 bits. Vous allez devoir faire vos propres fonctions si vous utilisez un format différent.
Commençons par voir nos constructeurs :

Constructeurs

```
BitmapFont::BitmapFont()
{
    //Met bitmap a NULL
    bitmap = NULL;
}

BitmapFont::BitmapFont( SDL_Surface *surface )
{
    //Construit le font
    build_font( surface );
}
```

Le premier, sans paramètres, met simplement le pointeur vers le bitmap à *NULL*.
Le second construit notre font avec les variables passées en paramètres en utilisant la méthode **build_font()**.
Pour construire notre font, on n'a pas besoin de beaucoup de chose : une surface. On pourrait éventuellement passer aussi la taille d'une cellule (hauteur et largeur).

build_font()

```
void BitmapFont::build_font( SDL_Surface *surface )
{
    //Si la surface est NULL
    if( surface == NULL )
    {
        return;
    }

    //Recupere le bitmap
    bitmap = surface;

    //Mise en place de la couleur du background (fond)
    Uint32 bgColor = SDL_MapRGB( bitmap->format, 0, 0xFF, 0xFF );

    //Dimension des cellules
    int cellW = bitmap->w / 16;
    int cellH = bitmap->h / 16;
```

La première chose à faire est de vérifier la validité de la surface passée en paramètre.
La suite est très simple : on assigne la surface comme étant notre bitmap, on définit la couleur de fond ainsi que la taille des cellules.
Une feuille de font bitmap typique fait 256 caractères dans l'ordre ASCII. Chaque cellule contenant une lettre faisant la même taille. Nous avons donc 16 lignes et 16 colonnes et donc la hauteur d'une cellule est égale à la hauteur de notre surface divisé par 16 (de même pour la largeur d'une cellule).
Maintenant qu'on a notre bitmap et qu'on a définit les cellules, il est temps de découper individuellement les lettres.

Découpage - partie 1

```
    //Le caractere courant
    int currentChar = 0;

    //On parcours les lignes des cellules
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Découpage - partie 1
```
    for( int rows = 0; rows < 16; rows++ )
    {
        //On parcours les colonnes des cellules
        for( int cols = 0; cols < 16; cols++ )
  {
```

Dans un premier temps, on définit un entier qui nous permettra de garder une trace du caractère qu'on est en train de construire. Ensuite on commence à parcourir notre font bitmap cellule par cellule.

Découpage - partie 2
```
            //caractere courant
            chars[ currentChar ].x = cellW * cols;
            chars[ currentChar ].y = cellH * rows;

            //Mise en place des dimensions du caractere
            chars[ currentChar ].w = cellW;
            chars[ currentChar ].h = cellH;
```

Le découpage de chaque caractère est classique avec la définition de la position en x/y et la taille en hauteur/largeur. Là où ça va se compliquer un petit peu c'est qu'on va essayer de découper le caractère au plus proche plutôt que de le découper avec la taille de la cellule. En effet, chaque caractère ne faisant pas la même taille, en découpant avec la taille de la cellule, on se retrouvera rapidement avec des espaces entre chaque lettre complètement chaotiques. On va donc découper plus proprement chaque caractère en parcourant les pixels de chacune de nos cellules.

Découpage - partie 3
```
            //On parcours les colonnes des pixels
            for( int pCol = 0; pCol < cellW; pCol++ )
            {
                //On parcours les lignes des pixels
                for( int pRow = 0; pRow < cellH; pRow++ )
                {
                    //Recupere les coordonnees du pixel
                    int pX = ( cellW * cols ) + pCol;
                    int pY = ( cellH * rows ) + pRow;
```

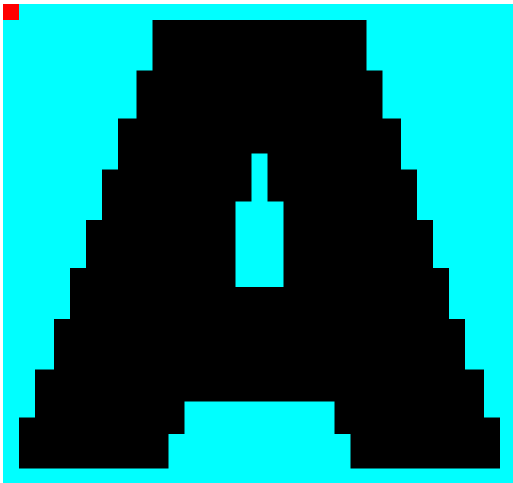
On commence par le parcours des colonnes afin de couper en largeur. On calcul les coordonnées en ajoutant les coordonnées relatives du pixel aux coordonnées de la cellule courante.

Découpage - partie 4
```
                    //Si un pixel non "colorkey" est trouve
                    if( get_pixel32( pX, pY, bitmap ) != bgColor )
                    {
                        //Coordonnee x
                        chars[ currentChar ].x = pX;

                        //On arrete la boucle
                        pCol = cellW;
                        pRow = cellH;
                    }
                }
            }
```

Le scan de notre cellule ressemble donc à ça :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Searching...



On récupère la couleur du pixel et si cette couleur est différente du fond définit, alors on a trouvé le début de notre caractère.
On met donc à jour la position x de notre caractère avec la position du pixel.
On fait la même chose de l'autre côté de la cellule afin de récupérer la largeur de notre caractère.

Découpage - partie 5

```
            //On parcours les colonnes des pixels
            for( int pCol_w = cellW - 1; pCol_w >= 0; pCol_w-- )
            {
                //On parcours les lignes des pixels
                for( int pRow_w = 0; pRow_w < cellH; pRow_w++ )
                {
                    //Recupere les coordonnees du pixel
                    int pX = ( cellW * cols ) + pCol_w;
                    int pY = ( cellH * rows ) + pRow_w;

                    //Si un pixel non "colorkey" est trouve
                    if( get_pixel32( pX, pY, bitmap ) != bgColor )
                    {
                        //longueur du caractere courant
                        chars[ currentChar ].w = ( pX - chars[ currentChar ].x ) + 1;

                        //On arrete la boucle
                        pCol_w = -1;
                        pRow_w = cellH;
                    }
                }
            }
```

On démarre donc cette fois ci de la droite de la cellule et on remonte en arrière colonne par colonne.
Une fois un des pixels du caractère trouvé, la taille est calculée en soustrayant à la coordonnée x de notre pixel la position x de notre caractère déjà défini. On fini par ajouter 1 pour prendre en compte le pixel.
Voila, on a défini la taille d'un caractère, on passe donc au suivant et ainsi de suite.
N'oubliez pas que ce qu'on a fait ici ne fonctionne qu'avec un font bitmap ayant 256 caractères dans l'ordre ASCII placés dans 16 lignes et 16 colonnes. Il faudra faire autrement pour d'autres styles de font bitmap.
Vous remarquerez que nous n'avons pas découpé plus en détail la hauteur de nos caractères. C'est parce que nous avons besoin, pour la hauteur, d'une taille uniforme déjà offerte par la hauteur de la cellule afin d'assurer les espaces entre deux lignes.
Il nous reste à voir la méthode show_text avant de s'attaquer à l'application de notre classe.

show_text()

```
void BitmapFont::show_text( int x, int y, std::string text, SDL_Surface *surface )
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

show_text()
```
{
    //coordonnees temporaires
    int X = x, Y = y;
```

Cette fonction va nous permettre d'afficher le texte.
La première chose qu'on fait est de conserver les coordonnées passées en paramètre dans des variables temporaires. Les coordonnées passées en paramètre vont nous servir de coordonnées de base et les coordonnées temporaires vont nous servir pour déterminer où sera posé le prochain caractère.

show_text()
```
    //verification que le font a bien ete construit
    if( bitmap != NULL )
    {
        //On parcours le texte
        for( int show = 0; text[ show ] != '\0'; show++ )
        {
```

On vérifie que le bitmap a bien été construit puis on va parcourir le texte.

show_text()
```
            //Si le caractere courant est un espace
            if( text[ show ] == ' ' )
            {
                //On bouge de la taille d'un caractere
                X += bitmap->w / 32;
            }
            //Si le caractere courant est un "newline"
            else if( text[ show ] == '\n' )
            {
                //On descent
                Y += bitmap->h / 16;

                //On revient en arriere
                X = x;
            }
```

On vérifie si le caractère est un espace ou un *newline* (nouvelle ligne).
Si le caractère est un espace, on bouge la coordonnée temporaire vers la droite. On a choisi ici de bouger de la taille d'une demi cellule.
Si le caractère est un *newline*, la coordonnée temporaire descend de la taille d'une cellule et on revient à la coordonnée x de base. On revient donc à la ligne.

show_text()
```
            else
            {
                //recupere la valeur ASCII du caractere
                int ascii = (int)text[ show ];

                //Affiche le caractere
                apply_surface( X, Y, bitmap, surface, &chars[ ascii ] );

                //On bouge de la longueur du caractere + un pas de un pixel
                X += chars[ ascii ].w + 1;
            }
        }
    }
}
```

Si le caractère courant n'est pas un caractère spécial, on applique le caractère sur notre fenêtre SDL.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Pour afficher le caractère, on récupère la valeur ASCII en castant notre caractère en *integer*. Ensuite on applique notre caractère à l'écran. **C'est là l'intérêt de mettre nos caractères dans l'ordre ASCII**.
Ensuite la coordonnée temporaire est bougée de la taille du caractère plus un. Ainsi, la caractère suivant sera appliqué juste à la suite de celui-ci avec seulement un espace d'un pixel.
La boucle continue jusqu'à ce qu'on ai plus aucun caractère à afficher.
Nous allons finir ce tutoriel en utilisant notre classe.

main

```
//Creation de notre font
    BitmapFont font( bitmapFont );

    //On remplit l'ecran de blanc
    SDL_FillRect( screen, &screen->clip_rect, SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF ) );

    //Affiche le texte
    font.show_text( 50, 50, "Bienvenue sur Developpez.com\n\nLoka", screen );

    //Mise à jour de l'écran
    if( SDL_Flip( screen ) == -1 )
    {
        return 1;
    }
```

Voici ce que ça donne au final (je rappelle que j'ai fais les caractères rapidement à la souris) :

Télécharger les sources du chapitre XVII (113 ko)

Version pdf (126 ko - 14 pages)

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Remerciements

<= Retour au chapitre XVI : Alpha Blending **-=Index Tutos SDL=-** **Accéder au chapitre XVIII : Thread =>**

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.