Tema 7: Aspectos avanzados de SDL - La biblioteca TTF

Tecnología de la Programación de Videojuegos 1

Grado en Desarrollo de Videojuegos

Curso 2023-2024

Miguel Gómez-Zamalloa Gil con cambios de Rubén Rubio Cuéllar

Departamento de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

La biblioteca TTF

- Permite usar fuentes TrueType en aplicaciones SDL
- TrueType es un formato estándar de tipos de letra escalables
  - Creado por Apple y Microsoft en los años 80
  - Ficheros con extensión `.ttf`
  - Elementos vectoriales que utilizan curvas de Bézier cuadráticas
  - Las funciones cuadráticas son convertidas a mapas de bits teniendo en cuenta el tamaño y otros aspectos
- Otros formatos más avanzados
  - OpenType (OTF), creado por Adobe y Microsoft
  - Apple Advanced Typography (AAT): extiende TTF con características similares a las de OTF

La biblioteca TTF

- Es una biblioteca externa y requiere instalación (diferente dependiendo el entorno)
- En nuestra plantilla de proyectos SDL
  - Copiar la biblioteca en el directorio de la solución (como está SDL_image)
  - Modificar el archivo de propiedades para que la incluya (ver README)
- Para usarla hay que incluir

```
#include <SDL_ttf.h>
```

- Las funciones, tipos y constantes empiezan por TTF_

Funciones básicas

1. Inicialización `TTF_Init()`
2. Carga de una fuente (dado un fichero .ttf y un tamaño en puntos)

```
TTF_Font* font = TTF_OpenFont(filename.c_str(), size);
```

3. Generación de textura a partir de una fuente y un texto

```
SDL_Surface* surf = TTF_RenderUTF8_Solid(font, text.c_str(), color);
SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, surf);
```

4. Renderizado (como cualquier otra textura)

```
SDL_RenderCopy(renderer, texture, &srcDest, &destRect);
```

5. Liberación de la fuente `TTF_CloseFont(font)`
6. Finalización `TTF_Quit()`

Integración con nuestras clases

Creamos una nueva clase Font

```
class Font {
    TTF_Font* font = nullptr;
public:
    Font() = default;
    Font(const std::string& filename, int size);
    Font(const Font& font) = delete;
    ~Font();

    void load(const std::string& filename, int size);
    void free();

    SDL_Surface* generateSurface(const string& text, SDL_Color color) const;
    Texture* generateTexture(SDL_Renderer* renderer, const string& text,
                             SDL_Color color) const;
};
```

Integración con nuestras clases

```
Font::Font(const string& filename, int size) { load(filename, size); }

Font::~Font() { free(); }

void Font::load(const string& filename, int size) {
    font = TTF_OpenFont(filename.c_str(), size);
    ...
}

void Font::free() {
    TTF_CloseFont(font);
    font = nullptr;
}

SDL_Surface* Font::generateSurface(const string& text, SDL_Color color) const {
    return TTF_RenderUTF8_Solid(font, text.c_str(), color);
}
```

Integración con nuestras clases

```
Texture* Font::generateTexture(SDL_Renderer* renderer, const string& text,
                               SDL_Color color)
{
   SDL_Surface* surface = generateSurface(text.c_str(), color);
   SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, surface);

   if (texture == nullptr)
      throw SDLError("al crear una textura a partir de texto");

   return new Texture(renderer, texture, 1, 1);
}
```

Más funciones

- Aparte de `TTF_RenderUTF8_Solid`, la biblioteca dispone de al menos otras 39 funciones para el renderizado de texto
  1. Cuatro tipos de renderizado: `Solid`, `Shaded`, `Blended`, `LCD`
  2. Cinco formatos de texto: `Glyph`, `Glyph32`, `Text`, `UTF8` y `UNICODE`
  3. Con ajuste de líneas o no

```
TTF_Render①_②(_Wrapped)(TTF_Font*, text, SDL_Color)
```

- Las tipografías tienen además una serie de propiedades y atributos que se pueden establecer y obtener
  - Por ejemplo, el estilo (normal, cursiva, negrita, subrayado):

```
TTF_SetFontStyle(font, TTF_STYLE_BOLD | TTF_STYLE_UNDERLINE);
```

- Más en www.sdltutorials.com/sdl-ttf y wiki.libsdl.org/SDL2_ttf

Cambiar el color del texto

Un mismo texto (o textura en general) se puede pintar en distintos colores si es blanca usando `TextureColorMod` (multiplica cada canal por un factor)

```
void
Texture::render(const SDL_Rect& rect, SDL_Color color) const
{
    SDL_SetTextureColorMod(texture, color.r, color.g, color.b);
    render(rect);

    // Restablece el color original de la textura
    SDL_SetTextureColorMod(texture, 255, 255, 255);
}
```