Lesson 07 – Text Rendering & Input

SDL_ttf

SDL_ttf provides an API for True Type Font (TTF) loading and rendering. It works very similarly to other extensions we've used. Like SDL_mixer, SDL_ttf introduces another asset structure; TTF_Font.

The setup process is the same as the other extensions we've used. The documentation can be found here.

Initialization

Again like the other extensions, SDL_ttf includes an initialization function, TTF_Init(). This function does not take any parameters; simply call it in your program startup.

```
if ( TTF_Init() < 0 ) {
        cout << "Error
```

Example Program

Download

```
#include <iostream>
#include <string>

#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>

using namespace std;

bool init();
void kill();
bool loop();

// Pointers to our window, renderer,
texture, and font
SDL_Window* window;
SDL_Renderer* renderer;
SDL_Texture *texture, *text;
TTF_Font* font;
string input;

int main(int argc, char** args) {

        if ( !init() ) {
                system("pause");
initializing SDL_ttf: " <<
TTF_GetError() << endl;
}
```

To load fonts, SDL_ttf provides the function TTF_OpenFont() for single font files and the function TTF_OpenFontIndex() to load a font from a file containing multiple. Like the other asset loading functions, TTF_OpenFont() takes a file name, but now also a font size in pixels. The function returns a pointer to a new TTF_Font or NULL on failure.

```
TTF_Font* font;

font =
TTF_OpenFont("font.ttf",
24);
if ( !font ) {
        cout << "Failed to load
font: " << TTF_GetError()
<< endl;
}
```

Rendering Text

There are actually quite a few ways to render text with SDL_ttf. You can render and store individual glyphs (characters), you can render UNICODE strings, you can render text in several different ways (solid, blended, shaded), and more. For now, we'll just go over a simple way to get text to the screen.

The basis of all the text rendering methods is creating a SDL_Surface that contains the rendered text

```
                return 1;
        }

        while ( loop() ) {
                // wait before processing
the next frame
                SDL_Delay(10);
        }

        kill();
        return 0;
}

bool loop() {

        static const unsigned char*
keys = SDL_GetKeyboardState(
NULL );

        SDL_Event e;
        SDL_Rect dest;

        // Clear the window to white
        SDL_SetRenderDrawColor(
renderer, 255, 255, 255, 255 );
        SDL_RenderClear( renderer );

        // Event loop
        while ( SDL_PollEvent( &e ) !=
0 ) {
                switch (e.type) {
                        case SDL_QUIT:
                                return false;
                        case
SDL_TEXTINPUT:
                                input +=
e.text.text;
                                break;
                        case
SDL_KEYDOWN:
                                if
(e.key.keysym.sym ==
SDLK_BACKSPACE && input.size())
{

input.pop_back();
                                }
                                break;
                }
        }
```

rendered text.
TTF_RenderText_Solid() is the most straightforward way to do this. This function simply takes a TTF_Font pointer, a c-string to render, and a SDL_Color to render the text in. It returns a new SDL_Surface, or NULL on failure.

```
SDL_Surface* text;
// Set color to black
SDL_Color color = { 0, 0, 0
};

text =
TTF_RenderText_Solid(
font, "Hello World!", color );
if ( !text ) {
	cout << "Failed to
render text: " <<
TTF_GetError() << endl;
}
```

Once you have a surface containing your rendered text, you can render it with the methods discussed in lesson 02 or 04. For example, you can create a texture from this surface and render it with SDL_RenderCopy().

```
SDL_Texture* text_texture;

text_texture =
SDL_CreateTextureFromSurfa
 renderer, text );

SDL_Rect dest = { 0, 0,
text->w, text->h };

SDL_RenderCopy(
renderer, text_texture,
&dest );
```

You might notice that with this method, we must completely re-render glyphs (a costly operation)

```
	// Render texture
	SDL_RenderCopy(renderer,
texture, NULL, NULL);

	SDL_Color foreground = { 0, 0,
0 };

	if ( input.size() ) {
		SDL_Surface* text_surf =
TTF_RenderText_Solid(font,
input.c_str(), foreground);
		text =
SDL_CreateTextureFromSurface(renderer,
 text_surf);

		dest.x = 320 - (text_surf->w
/ 2.0f);
		dest.y = 240;
		dest.w = text_surf->w;
		dest.h = text_surf->h;

SDL_RenderCopy(renderer, text,
NULL, &dest);

		SDL_DestroyTexture(text);

SDL_FreeSurface(text_surf);
	}

	// Update window
	SDL_RenderPresent( renderer
);

	return true;
}

bool init() {
	if ( SDL_Init(
SDL_INIT_EVERYTHING ) < 0 ) {
		cout << "Error initializing
SDL: " << SDL_GetError() << endl;
		return false;
	}

	if ( IMG_Init(IMG_INIT_PNG) <
0 ) {
		cout << "Error initializing
SDL_image: " << IMG_GetError() <<
endl;
		return false;
	}
```

whenever we want to change our output. This is pretty inefficient if you need to render changing text. To get around this, you can render individual glyphs to textures, which you then output in the correct positions based on what text you want to display. However, this is pretty complicated, so we won't get into it here. (However, another extension, SDL_FontCache can do this for you.)

When using this method, you will have to re-create your text (calling TTF_RenderText_Solid()) every time it changes. Remember to free/destroy any surfaces our textures you need to re-create.

```
SDL_DestroyTexture(
text_texture );
SDL_FreeSurface( text );
```

Text Input

While you can technically input text just by polling keyboard events, SDL provides a more convenient way to do this via SDL_TextInputEvent. This event type packages keyboard input in a more convenient way for text input; keys pressed are sent as a c-string.

SDL will not by default register these events; you must call SDL_StartTextInput() to enable them. SDL will also start sending SDL_TextEditingEvents, but you don't need to handle these to do basic text input. When you're

```
	// Initialize SDL_ttf
	if ( TTF_Init() < 0 ) {
		cout << "Error intializing
SDL_ttf: " << TTF_GetError() <<
endl;
		return false;
	}

	window = SDL_CreateWindow(
"Example",
SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,
640, 480, SDL_WINDOW_SHOWN );
	if ( !window ) {
		cout << "Error creating
window: " << SDL_GetError()  <<
endl;
		return false;
	}

	renderer =
SDL_CreateRenderer( window, -1,
SDL_RENDERER_ACCELERATED
);
	if ( !renderer ) {
		cout << "Error creating
renderer: " << SDL_GetError() <<
endl;
		return false;
	}

	SDL_Surface* buffer =
IMG_Load("test.png");
	if ( !buffer ) {
		cout << "Error loading
image test.png: " << SDL_GetError()
<< endl;
		return false;
	}

	texture =
SDL_CreateTextureFromSurface(
renderer, buffer );
	SDL_FreeSurface( buffer );
	buffer = NULL;
	if ( !texture ) {
		cout << "Error creating
texture: " << SDL_GetError() << endl;
		return false;
	}
```

done with text input, call SDL_StopTextInput() to disable these events.

The input characters are stored in the "text" member of SDL_TextInputEvent. You can capture this input by appending these characters to an input buffer string, and show the user what they're typing by displaying the buffer string.

Finally, you can still listen to keyboard events—for example, you can have the backspace key remove the last character from the buffer string. This is done in the example program.

```
SDL_StartTextInput();
string in;
bool running = true;

while ( running ) {
	SDL_Event ev;
	while (
SDL_PollEvent( &ev ) ) {
		if ( ev.type ==
SDL_TEXTINPUTEVENT ) {
			in +=
ev.text.text;
			cout << " > "
<< in << endl;
		} else if ( ev.type
== SDL_KEYDOWN &&
ev.key.keysym.sym ==
SDLK_BACKSPACE &&
in.size()) {

in.pop_back();
			cout << " > "
<< in << endl;
		} eles if ( ev.type
== SDL_QUIT ) {
			running =
false;
		}
	}
```

```
	// Load font
	font = TTF_OpenFont("font.ttf",
72);
	if ( !font ) {
		cout << "Error loading font:
" << TTF_GetError() << endl;
		return false;
	}

	// Start sending SDL_TextInput
events
	SDL_StartTextInput();

	return true;
}

void kill() {
	SDL_StopTextInput();

	TTF_CloseFont( font );
	SDL_DestroyTexture( texture );
	texture = NULL;

	SDL_DestroyRenderer(
renderer );
	SDL_DestroyWindow( window );
	window = NULL;
	renderer = NULL;

	TTF_Quit();
	IMG_Quit();
	SDL_Quit();
}
```

```
}

SDL_StopTextInput();
```

Shutdown

Shutting down SDL_ttf is just like the other extensions; free any loaded fonts with TTF_CloseFont() and call TTF_Quit().

```
TTF_CloseFont( font );
TTF_Quit();
```
