Lesson 04 – Geometry Rendering

Home

GPU Rendering | The Renderer | Clearing the Renderer | Draw Settings | Drawing | Shutdown

GPU Rendering

So far, you've been using software, or CPU, rendering. This means that to blit a surface to the window, your computer's CPU must iterate through each pixel, adjusting and copying the values.

Hardware, rendering, on the other hand, leverages your computer's GPU (Graphics Processing Unit). Hardware rendering can be an order of magnitude faster than software, as the GPU is optimized for these exact workloads. Instead of going through the pixel data individually, a GPU can preform calculations in parallel, massively increasing the throughput.

The Renderer

The SDL structure SDL_Renderer

Example Program

Download

```
#include <iostream>
#include <SDL.h>

using namespace std;

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

// Pointers to our window and
renderer
SDL_Window* window;
SDL_Renderer* renderer;

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

	if ( !init() ) return 1;

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

The SDL structure SDL_Renderer represents a *rendering context*. This means that it contains all current settings related to rendering, as well as the instructions for how to render the current frame. In your program, you will use functions such as SDL_SetRenderDrawColor() to change settings in the context, and functions such as SDL_RenderDrawPoint() to do rendering actions.

To create a rendering context, you can use the function

```
	kill();
	return 0;
}

bool loop() {

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

	SDL_Event e;
	SDL_Rect r;
	// For mouse rectangle (static to
presist between function calls)
	static int mx0 = -1, my0 = -1,
mx1 = -1, my1 = -1;

	// 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_MOUSEBUTTONDOWN:
				mx0 = e.button.x;
				my0 = e.button.y;
				break;
			case
SDL_MOUSEMOTION:
				mx1 = e.button.x;
				my1 = e.button.y;
				break;
			case
SDL_MOUSEBUTTONUP:
				mx0 = my0 =
mx1 = my1 = -1;
				break;
		}
	}

	// Set drawing color to black
	SDL_SetRenderDrawColor(
renderer, 0, 0, 0, 255 );

	// Test key states - this could
```

```
also be done with events
	if ( keys[SDL_SCANCODE_1] )
{
		SDL_RenderDrawPoint(
renderer, 10, 10 );
	}
	if ( keys[SDL_SCANCODE_2] )
{
		SDL_RenderDrawLine(
renderer, 10, 20, 10, 100 );
	}
	if ( keys[SDL_SCANCODE_3] )
{
		r.x = 20;
		r.y = 20;
		r.w = 100;
		r.h = 100;
		SDL_RenderFillRect(
renderer, &r );
	}

	// Render mouse rectangle
	if ( mx0 != -1 ) {
		r.x = mx0;
		r.y = my0;
		r.w = mx1 - mx0;
		r.h = my1 - my0;
		SDL_RenderDrawRect(
renderer, &r );
	}

	// Update window
	SDL_RenderPresent( renderer
);

	return true;
}

bool init() {
	// See last example for
comments
	if ( SDL_Init(
SDL_INIT_EVERYTHING ) < 0 ) {
		cout << "Error initializing
SDL: " << SDL_GetError() << endl;
		system("pause");
		return false;
	}

	window = SDL_CreateWindow(
"Example",
```

```
SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,
150, 150, SDL_WINDOW_SHOWN );
	if ( !window ) {
		cout << "Error creating
window: " << SDL_GetError()  <<
endl;
		system("pause");
		return false;
	}

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

	SDL_SetRenderDrawColor(
renderer, 255, 255, 255, 255 );
	SDL_RenderClear( renderer );
	return true;
}

void kill() {
	// Quit
	SDL_DestroyRenderer(
renderer );
	SDL_DestroyWindow( window );
	SDL_Quit();
}
```

SDL_CreateWindowAndRenderer() or SDL_CreateRenderer(). The former does what you'd expect: it creates both a window and renderer in conjunction. The latter requires you to first create a window (as the window must be a parameter), but allows a bit more fine control over the initialization.

```
SDL_Window* window;
SDL_Renderer* renderer;

int result = SDL_CreateWindowAndRenderer( 640, 480, NULL,
&window, &renderer );
if ( result != 0 ) {
	cout << "Failed to create window and renderer: " <<
SDL_GetError() << endl;
```

```
}
```

We no longer use a surface to represent the window. Instead, the rendering context draws to a backbuffer, which is then shown on the window at the end of each frame. Hence, you can no longer call SDL_UpdateWindowSurface(). Instead, you use SDL_RenderPresent(). This function tells the renderer show its operations on its window.

```
SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255 );
SDL_RenderDrawPoint( renderer, 100, 100 );

// Update window
SDL_RenderPresent( renderer );
```

Clearing the Renderer

Clearing the renderer resets the window to a color—this is equivalent to blitting a rectangle over the whole window. You should clear the window after each frame, as otherwise drawing will persist between frames, even if moved.

As you'd expect, the function SDL_RenderClear() is used for this purpose. The only parameter is renderer; the associated window will be cleared to the current color.

```
SDL_RenderClear( renderer );

// do drawing

SDL_RenderPresent( renderer );
```

Draw Settings

The drawing state is contained in the rendering context, so SDL provides several functions to adjust it. To change the drawing color (effecting drawing functions and SDL_RenderClear()), use the function SDL_SetRenderDrawColor(). Its use is very straightforward—simply pass the R, G, B, and A color values. To get the current color, use the function SDL_GetRenderDrawColor().

```
SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255 ); // Set color to solid white

SDL_RenderClear( renderer ); // Clear the screen to solid white
```

To set the drawing blend mode, use the function SDL_SetRenderDrawBlendMode(). A blendmode is simply a method of drawing a texture with alpha (transparency) data—for example, the default blendmode simply blends the colors together based on the alpha value. However, SDL also supports additive and modulated blending, which greatly effect the final image. I encourage you to test these out on your own. To get the current blendmode, use the function SDL_GetRenderDrawBlendMode().

```
SDL_SetRenderDrawBlendMode( renderer, SDL_BLENDMODE_ADD ); // Switch to additive blending

SDL_RenderDrawLine( renderer, 0, 0, 100, 100 ); // Draw with additive blending
```

These settings and more are categorized under CategoryRender on the SDL wiki.

Drawing

As I've mentioned throughout this section, SDL provides several functions for drawing rudimentary shapes—namely, points, lines, rectangles, and filled rectangles. The functions are all used as you'd expect: the point and line functions simply take the screen coordinates of the shape, and the rectangle functions take a SDL_Rect with the same data.

- SDL_RenderDrawPoint()
- SDL_RenderDrawPoints()
- SDL_RenderDrawLine()
- SDL_RenderDrawLines()
- SDL_RenderDrawRect()
- SDL_RenderDrawRects()
- SDL_RenderFillRect()
- SDL_RenderFillRects()

All of these functions and more are categorized under CategoryRender on the SDL wiki.

```
SDL_SetRenderDrawColor( renderer, 0, 0, 255, 255 ); // Draw in solid blue

SDL_RenderDrawLine( renderer, 10, 10, 50, 25 ); // Draw a line

SDL_Rect r;
r.x = 150;
r.y = 25;
r.h = 75;
r.w = 120;
```

```
SDL_RenderFillRect( renderer, &r ); // Draw a filled rectangle

// etc
```

Shutdown

Shutting down is almost exactly the same as with software rendering, except now you must use the function SDL_DestroyRenderer() to free the renderer before shutting down the window. You still free surfaces in the same way.

```
SDL_DestroyRenderer( renderer );
SDL_DestroyWindow( window );

renderer = NULL;
window = NULL;
```

Made by Maxwell Slater © 2015–2017 | Contact me at mslater@nevada.unr.edu |

View this project on GitHub