Comparison of Rendering Performance Between Multimedia Libraries Allegro, SDL and SFML

KRISTOFFER ALMROTH
ANTON LE CLERCQ
Comparison of Rendering Performance Between Multimedia Libraries Allegro, SDL and SFML

KRISTOFFER ALMROTH

ANTON LE CLERCQ
Abstract

In this report the rendering performances of the multimedia libraries Allegro, SDL and SFML have been compared. Highest performance is achieved by writing code directly to the low level graphical APIs, though it requires much more work than using the multimedia libraries graphical functions built on one of these graphical APIs. Thus it is common to use a multimedia library or similar for visualization tasks. The total number of frames rendered in one second was counted for static, alpha blended, rotating, and moving images in each library. Every test was run with few to very many images, and the programs were tested on six different computers: three laptops with integrated GPUs and low power dual core CPUs, and three desktop computers with external GPUs and quad core CPUs with unlocked clock rate. Allegro performed best of the three on laptops when the image load was very high, but fell behind by up to 50% in all other cases. SDL had the strongest performance on desktop computers, especially when rendering very many images, making it a prime candidate for high load graphical applications on desktops. SFML performed best overall, making it the best choice when targeting a wide range of different machines.
Sammanfattning

I denna rapport jämförs renderingsprestandan mellan multimediabiblioteken Allegro, SDL och SFML. Den högsta prestandan uppnås genom att skriva kod direkt till en lågnivå-API för grafik, men det kräver mycket mer kod än att använda ett multimediabibliotek. Därför är det vanligt att använda ett multimediabibliotek eller något med liknande funktioner för visualiseringsarbeten. Jämförelsen bestod av att räkna det totala antalet skärmbilder som renderades under en sekund för statiska, semitransparenta, rotarande och rörliga bilder. Varje test kördes med 50 till 10 000 bilder som renderades samtidigt, och programmen testades på sex olika datorer, tre bärbara med integrerade GPUs och tvåkärniga energieffektiva CPUs, och tre stationära med externa GPUs och fyrkärniga CPUs med upplåst klockfrekvens. Allegro presterade bäst på bärbara datorer under en hög belastning, men var upp till 50% sämre i alla övriga tester. SDL presterade bäst på stationära datorer, därför är det ett bra val för krävande grafiska program på stationära datorer. SFML presterade bäst överlag, vilket gör det till det bästa valet för att skapa grafiska program som är tänkta att köras på olika starka datorer.
Contents

1 Introduction ................................................... 1
   1.1 Purpose ................................................. 2
   1.2 Research Question ..................................... 2
   1.3 Scope .................................................. 3
   1.4 Project Outline ........................................ 3

2 Background .................................................. 4
   2.1 Definitions ............................................. 4
   2.2 Graphics APIs .......................................... 5
       2.2.1 OpenGL ........................................... 6
       2.2.2 Direct3D ........................................... 6
   2.3 Multimedia libraries .................................... 6
       2.3.1 Allegro ............................................ 7
       2.3.2 SDL ............................................... 7
       2.3.3 SFML ............................................. 8
   2.4 Previous Research ...................................... 9

3 Method ....................................................... 11
   3.1 Dependencies .......................................... 11
   3.2 Test Layout ............................................ 12
   3.3 Test Images ............................................ 13
   3.4 Further Details ........................................ 13
   3.5 Rendering Implementation ............................. 14
       3.5.1 Allegro ............................................ 15
       3.5.2 SDL ............................................... 15
       3.5.3 SFML ............................................. 15
   3.6 Tested Computers ....................................... 16
       3.6.1 Desktop Computers ............................... 16
       3.6.2 Laptop Computers ................................. 16
4 Results

4.1 General Test Information ........................................ 17
4.2 Static Sprites ..................................................... 18
  4.2.1 Desktop Computer ........................................ 18
  4.2.2 Laptop ..................................................... 19
4.3 Alpha Blended Sprites ........................................... 20
  4.3.1 Desktop Computer ........................................ 20
  4.3.2 Laptop ..................................................... 21
4.4 Rotating Sprites .................................................. 22
  4.4.1 Desktop Computer ........................................ 22
  4.4.2 Laptop ..................................................... 23
4.5 Moving Sprites ................................................... 24
  4.5.1 Desktop Computer ........................................ 24
  4.5.2 Laptop ..................................................... 25
4.6 Result Summary ................................................... 26
  4.6.1 Desktop Computers ........................................ 26
  4.6.2 Laptop Computers ......................................... 27

5 Discussion ................................................................... 29

5.1 Differences in Performance ....................................... 29
5.2 Reasons for the Performance Difference ....................... 31
5.3 Potential Sources of Error ....................................... 32

6 Conclusion .................................................................. 34

6.1 Future Research ..................................................... 35

Bibliography .................................................................. 36

A Tested Computers ...................................................... 39

A.1 Computer 1 (Desktop) ............................................. 39
A.2 Computer 2 (Desktop) ............................................. 40
A.3 Computer 3 (Desktop) ............................................. 40
A.4 Computer 4 (Laptop) ................................................ 40
A.5 Computer 5 (Laptop) ................................................ 41
A.6 Computer 6 (Laptop) ................................................ 41
Chapter 1

Introduction

Rendering is a staple in much of today’s computer use. From simple websites to complex simulations, designers and programmers often want to show information visually on a screen. Graphical processing units (GPU), specialized hardware for handling graphics, are commonly in use in today’s computers to accelerate the process of rendering images to the screen. However, these GPUs may have wildly different syntaxes for how to communicate with them, largely depending on the operating system. To bridge the gap between software and hardware, a specification may be used. A specification, or API, acts as a recipe for how the code can contact and use the hardware. Two common such graphic specifications are Direct3D, which is Windows-specific, and OpenGL, which has support on all major operating systems.

A common practice among programmers is to use another layer of abstraction over these graphic APIs, to make the code more portable as well as easier to write. One type of such abstraction are multimedia libraries. A wealth of popular multimedia libraries are available. Three of the most common ones for several languages, including Java, Pyton, C++, and C, are SFML, SDL and Allegro (see table 1.1).

<table>
<thead>
<tr>
<th>Library</th>
<th>Language</th>
<th>API Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFML</td>
<td>C++11</td>
<td>OpenGL</td>
</tr>
<tr>
<td>SDL</td>
<td>C</td>
<td>OpenGL or DirectX</td>
</tr>
<tr>
<td>Allegro</td>
<td>C</td>
<td>OpenGL or DirectX</td>
</tr>
</tbody>
</table>

Table 1.1: Basic attributes of three multimedia libraries
1.1 Purpose

Multimedia libraries are commonly used for visualization, especially tasks where additional tools, such as user input or networking, are needed. For visualization, writing low level code directly to OpenGL or Direct3D has the potential to result in a more efficient finished application but it is a much bigger task to undertake compared to the easy to use interface for writing graphics in the multimedia libraries. Thus, when choosing multimedia library, one important factor is the performance of the graphics library.

While relevant to assess what the intended use of the graphical library is (i.e. what it is optimized for), Allegro, SDL, and SFML seem to occupy more or less the same niche, being multimedia libraries commonly used for 2D games and applications. Therefore no conclusions can be drawn on which should be most optimized for which situation.

When comparing multimedia libraries graphical performance, it is interesting to look at some of the most common graphical usages, which apart from drawing static images, include alpha blended (semi-opaque), rotating and moving sprites. The graphical performance can be measured by the frame rate of the program, if the frame rate is unlocked. The operating system will not naturally influence the resolution or graphical fidelity of the image, which means more complex images will simply take longer to render. Because of this, more efficient rendering will directly lead to more frames rendered per second.

Based on these qualities, it might be relevant to consider what libraries has the best rendering performance in different situations, so that developers can choose a multimedia library that will work well for their project. It is desirable to find the optimal library at the start of a project, to avoid costly and time-consuming rewrites of basic systems.

1.2 Research Question

*How do the three multimedia libraries SDL, SFML and Allegro compare when it comes to graphical visualization in terms of frame rate?*
1.3 Scope

This research project compares three popular cross-platform multimedia libraries in terms of graphics efficiency when rendering two-dimensional images. The study is limited to analyzing static images, alpha blended images, rotating images and moving images on the Windows operating system, and only a single image (rendered multiple times). Three equivalent programs will be created in C++, one for each library to be tested, to measure the performance in each of the four situations. Future research could improve upon this report by adding more comparisons in common graphical operations, such as shaders, as well as testing other operating systems, for example Linux and MacOS.

1.4 Project Outline

The background contains the core concept of graphics programming, starting with the low level graphics APIs OpenGL and DirectX, then going one further abstraction level to the three multimedia libraries Allegro, SDL, and SFML. The next part of the background contains prior research into how the three multimedia libraries compare when it comes to graphical performance.

All tools, frameworks, and hardware used are described in the method chapter. The chapter also goes through the design of the program used to perform all tests. The following chapter contains all results from the multimedia library comparison, visualized in several diagrams. Those results are then discussed in the next chapter, ending with our conclusion.
Chapter 2

Background

This chapter contains the following general subjects:

- Definitions of important concepts
- Low level graphics APIs
- Multimedia libraries
- Previous comparisons between the multimedia libraries

2.1 Definitions

- **API**: Application programming interface, a set of clearly defined methods for communication between components.
- **CPU**: Central processing unit
- **GPU**: Graphical processing unit
- **Alpha blending**: A way to display an image that has semi-transparent or transparent pixels. Each pixel normally has three different colors, red, blue and green. An 8-bit color representation has a value of 0 to 255 for each of the colors. With alpha blending, a fourth value is added to each pixel, the alpha channel. The alpha channel usually has the same number of bits as the color representation, which is 256 different levels in the 8-bit representation. The alpha
channel ranges from 0 which is fully transparent to 255 which is fully opaque \[1\].

- **Hardware acceleration:** Offload a certain task from the CPU to hardware that is specially made to perform that type of tasks \[2\]. In this report, hardware acceleration will mainly refer to utilizing the GPU for more efficient rendering.

- **RAM:** Random access memory, a type of memory where each byte can be read or written to with nearly the same delay independent of the physical location of the data. In this report, RAM will refer to DRAM, Dynamic Random Access Memory, a special type of RAM often used as the main memory for a computer \[3\].

- **VRAM:** Video RAM, acts as a buffer between the display and the graphics card. VRAM can be accessed simultaneously by the graphics card and the display, allowing the graphics card to process the next frame at the same time as the display reads a previous one \[4\]. VRAM is used by the graphics card. If there is no dedicated GPU in the system, a common solution is to use hardware acceleration simply using the regular system memory \[5\].

- **Vertex Array:** A vertex is a graphical entity containing among other things a position and a color. A vertex array groups vertices in a way that the GPU can decode and render it \[6\].

- **Sprite:** Broadly synonymous with 'interactive image' in modern computer game parlance, a sprite is simply an image with a certain position and size that can be manipulated \[7\]. In the context of this report, 'sprite' will be used synonymously with 'image'.

### 2.2 Graphics APIs

OpenGL and Direct3D are two of the most common graphics APIs used for hardware accelerated 3D and 2D graphical applications. Window handling and other similar features are handled by the operating system, not the graphics APIs. Apart from different implementation on some of the core features, the functionality of the two APIs are nearly identical.
2.2.1 OpenGL

OpenGL is a widely used specification for rasterization-based graphical rendering. It does not provide anything in the way of animations, image file processing, etc. In fact, it is not actually a software, but rather a standard for how the rendering should be handled, where several different implementations exist. Hardware vendors are responsible for making their own implementations, and these so-called 'drivers' translate the OpenGL API calls to GPU commands. OpenGL is developed by Khronos. All major operating systems support OpenGL, including Windows, MacOS and Linux. The current version is 4.6 [8].

OpenGL acts as a state machine, where the state is managed using OpenGL Objects. The OpenGL context allocates the memory used by the OpenGL Objects. Buffered Objects and Vertex Arrays are two different types of OpenGL Objects [9].

2.2.2 Direct3D

Direct3D is one of the main competitors to OpenGL on the Windows operating system, developed by Microsoft. Direct3D provides an API for highly parallelised rendering calls to the GPU. Direct3D is only available on Microsoft products, including Windows and Xbox. It is one of the components in the DirectX suite. The latest version of DirectX is 12 [10].

Compared to OpenGL, Direct3D does not handle the hardware resources for the GPU on its own, instead the application is supposed to handle it using DXGI. Thus, Direct3D is more flexible when it comes to resource management, allowing the developer to allocate the memory in the most efficient way possible. This flexibility also makes Direct3D more complicated to work with [11].

2.3 Multimedia libraries

Due to the relatively low-level functionality of the graphics APIs, there are a multitude of multimedia libraries with their own rendering functions. In a programming context, multimedia libraries consist of code
that handles or help with the handling of graphics, audio, and user inputs. Three such libraries are SFML (*Simple and Fast Multimedia Library*), SDL (*Simple DirectMedia Layer*), and Allegro (originally from *Atari Low-Level Game Routines*). They all have their own custom-implemented drawing methods that use OpenGL or Direct3D depending on the operating system, but are one further abstraction level away from the raw graphic APIs.

### 2.3.1 Allegro

Allegro is an open source multimedia library available on Windows, Linux, MacOS, Android and iOS. Initially released in early 1990, it is developed in the C language and according to the C99 standard [12]. Allegro works natively with C (and thus C++), with unofficial language bindings for several of the most popular programming languages, including Java, Go and Python [13].

OpenGL and DirectX is used for rendering objects to screen, depending on operating system. On windows, DirectX is used. The graphics library is purely 2D, with hardware acceleration support. It also has support for video playback, audio recording and fonts. Allegro is not in itself object oriented since it is built using C.

In Allegro, images are saved as bitmap structures in the RAM, i.e. a packed array of 1- to 4-byte integer-type numbers representing individual pixels of rectangular images. Those bitmaps are written to a special "screen" bitmap, which then renders on the screen [14].

### 2.3.2 SDL

Initially released in 1998, SDL is a open-source C-based multimedia library used in many commercial games, including a large portion of Valve-created games such as Half Life and Dota 2 [15]. It’s available for Windows, MacOS, Linux, iOS, and Android, as well as some more uncommon systems such as Raspberry Pi. Due to being written in C, SDL works natively on C++ as well, and there are bindings available for nine other languages, including D, Go, Pascal, and Python [16].
SDL is divided into multiple subsystems or categories, each with several header files. The general categories are Basics (for initialization and errors), Video (including both drawing and window handling), Input Events (both indirect and direct), Force Feedback (for haptic controllers), Audio, Threads, Timers, File Abstraction, Shared Object Support (a sort of alternative library), Platform and CPU Information, Power Management, and Additional Functionality (mostly platform-specific) [17].

The graphical section can use both DirectX and OpenGL, defaulting to DirectX on Windows. Rendering is done using SDL_Textures, which is a driver specific representation of the pixel data. SDL_Textures uses VRAM for its data if there exists a dedicated GPU [18]. SDL_Texture is rendered to SDL_Renderer, which is later rendering to the window.

### 2.3.3 SFML

SFML is a cross-platform, open source multimedia library originally released August 9, 2007 [19]. It works natively on Windows, MacOS and Linux, with experimental ports to Android and iOS [20]. SFML is an object oriented framework developed using C++ 11, with official bindings in C and .NET. Unofficial bindings includes D, Go, Java, Haskell and Python [21].

SFML is divided into five different modules: System, Window, Graphics, Audio and Network. SFML System is the base module, including timers, vector classes and several conversion functions. Window is a module that provides a rendering window, as well as event handling and input handling, including keyboard, mouse and controllers [22]. The Graphics module is purely 2D, with sprites, fonts and different shapes [23]. Audio lets the user load and play sound files, and stream larger music files [24]. The last module, Network, is made for socket-based communication utilizing HTTP and FTP [25].

The Graphics module is implemented using OpenGL, regardless of operating system. SFML uses sf::Texture to load images, a handle managed by OpenGL that points to the position in graphics memory where the image data is stored. Each pixel is stored as four bytes of data, one for each color channel (red, blue green) and one for the alpha channel. Transformations on that image is later stored in sf::Sprite, which points towards the texture. Transformations include scaling, rotation,
and color shifting [26].

When rendering a high number of sprites to screen in each frame, it is not recommended to use individual draw calls for each of those sprites. One possible solution to this problem is using OpenGL based vertex arrays to merge all sprites, then use only one draw call on the vertex array [26].

### 2.4 Previous Research

The earliest comparison between SFML and SDL is from 2007, done by Laurent Gomila, the creator of SFML[27]. The comparison was made August 28, 2007, only 19 days after the initial release of SFML 1.0 on August 9, 2007. It consisted of counting the number of frames rendered during five seconds for five different tests, of which the three tests relevant to this study can be found in table 2.1. It was tested on two computers, one "high-end" (AMD Dual Core 2 GHz processor, 1 GB RAM, GeForce 7800 GT graphics card), and one "low-end" (Intel Pentium Single-core 1.73 GHz processor, 256 MB RAM, integrated graphics).

1. Displaying 2000 static sprites
2. Displaying 2000 alpha-blended static sprites
3. Displaying 2000 rotating sprites

<table>
<thead>
<tr>
<th>Rendered</th>
<th>High-end computer</th>
<th>Low-end computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000 static sprites</td>
<td>9 frames</td>
<td>8 frames</td>
</tr>
<tr>
<td>2000 alpha-blended</td>
<td>5 frames</td>
<td>5 frames</td>
</tr>
<tr>
<td>2000 rotating sprites</td>
<td>3 frames</td>
<td>2 frames</td>
</tr>
</tbody>
</table>

Gomila explains that the results were heavily in favor for SFML since SDL at the time did not utilize hardware acceleration for image rendering. Instead the programmer was supposed to write OpenGL code directly into their SDL program to access that feature. It is also pointed out in several comments that the code was not optimised on the side of SDL, since Gomila was not used to writing code in the framework. After adding the missing optimizations, SFML was still faster, but not by the extreme margin in the test above.
Six years later, in 2013, the user *swarminglogic* updated the tests to the new version of SFML, 2.0, and the new version of SDL, 2.0. *Swarminglogic* points out that his tests use two different timers between the multimedia libraries, and that he does not remove the startup cost. The results of that test can be seen in table 2.2.

<table>
<thead>
<tr>
<th>Rendered</th>
<th>SDL</th>
<th>SFML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static sprites</td>
<td>903 frames</td>
<td>890 frames</td>
</tr>
<tr>
<td>Alpha-blended</td>
<td>576 frames</td>
<td>575 frames</td>
</tr>
<tr>
<td>Rotating sprites</td>
<td>869 frames</td>
<td>873 frames</td>
</tr>
</tbody>
</table>

Table 2.2: *Swarminglogic’s* 2013 comparison

Version 2.0 of SDL has hardware acceleration, making the comparison much more even. This time around, the graphical efficiency between the two multimedia libraries was nearly identical, differing with only one percent at most.
Chapter 3

Method

This chapter describes how the study was conducted, and with what materials. One of the primary candidates for measuring the efficiency of a multimedia library’s rendering implementation is the program’s speed. A simple measure of a program’s rendering speed is the number of frames it can render in a given unit of time (assuming that the frame rate is not fixed or capped). A higher number of frames means the program is faster at rendering a single frame.[28]

Directly equivalent test code was written for each of the three frameworks. See github for source code: https://github.com/antonlc/multimedia-library-comparison

3.1 Dependencies

The following programs were used to conduct the tests:

- SFML 2.5.0
- SDL 2.0.9 with SDL.Image 2.0.4 (to load PNG images for SDL)
- Allegro 5.2.4.0
- MinGW 7.3.0 (a C++ compiler)

Each of the libraries were used in their most up-to-date full release. The MinGW 7.3.0 compiler was selected since SFML required that specific
version of the compiler to work as intended, and it doesn’t affect the performance of Allegro or SFML.

3.2 Test Layout

To measure the rendering efficiency of the different multimedia libraries, several test cases needed to be made. The most simple type of graphical application is a window with static images. Another type of graphical application that is more advanced to implement is alpha blended images. It would not yield that much useful data only analyzing static images, therefore dynamic images are also analyzed, including images that rotate or move each frame, two common graphical tasks. In total, the following rendering operations were tested:

- Static images
- Alpha-blended images
- Rotating images
- Moving images

The number of images on screen for a normal program can vary drastically. To ensure useful results, good benchmarks need to be used for workload, that correctly represent real-world levels of strain [29]. A small program could use a dozen images rendered to screen at a time, while more demanding graphical application can reach into the thousands of images. To make the measurements of the multimedia libraries as relevant as possible, a big range of different number of sprites were tested. The number of simultaneous sprites (active images) were as follows over the eight tests: 50, 100, 200, 500, 1000, 2000, 5000, 10000.

Each run of testing went through all four of the rendering operations, measuring all eight of the sprite loads in one operation before continuing to the next one. This 32-test run was repeated 20 times per multimedia library ($n = 20$), resulting in 1920 data points per tested computer.
3.3 Test Images

Creating 10,000 different images for testing is unrealistic. Instead, testing usually uses a simplified input where key parameters have been identified [30]. The time it takes to render an image depends mostly on its resolution and complexity, not its form or color. Thus, it is sufficient to only have one variant of the image that is rendered on screen. The size of each image is in this case 128 by 128 pixels. The image used can be seen in figure 3.1.

Alpha blending is a special test case since the efficiency of alpha blending relies on the color of both the image that should be transparent and the background it should be blended into. To accurately test this, the image in Image 1 is colorful. For the first test cases with a low number of images, the probability that they overlap is low which results in most images being blended into the default background color of the multimedia library which likely is white. To solve this problem, an equally colorful background was created that all images could be blended into. The background can be seen in figure 3.2.

3.4 Further Details

These images need to be given a position that is identical between the three multimedia libraries. The C library stdlib was used with the random function rand, given the same seed between the three applications. The C pseudo random number generator has some problems, including
that lower numbers are slightly more likely to be generated \cite{31}. For this project, the highest randomness is not needed, since normal graphical applications usually does not have an even distribution of the images on screen. It is thus acceptable to use the C pseudo random number function.

The research study aims to measure the efficiency in rendering images, not to measure the time it takes in different multimedia libraries to start up the application or to load images. Thus, only the image rendering time is measured. Each test, consisting of one of the four rendering operations and one set of images, is ran during one second, where each frame is counted during that time. That test is repeated ten times for more accurate results.

All three multimedia libraries has their own timer implementations. For accurate tests, the same timer were used for all three applications. To avoid having to choose one timer from the multimedia libraries, a C++ standard library timer was chosen instead, from the \texttt{chrono} library.

3.5 Rendering Implementation

Generally, when rendering a large number of sprites to screen, it is not recommended to repeatedly call the rendering function in the multimedia libraries. Instead, all images should be grouped into a vertex array or equivalent data structure to amount to only one draw call each frame, as
they are effectively the same image, though it works for sprites that are static relative to each other. This study is not meant to compare batching techniques between OpenGL and Direct3D, instead it is meant to compare the standard rendering functions of all three multimedia libraries. Thus, it is still relevant to use the standard sprite data structure and its corresponding rendering call. If the multimedia library has a build in render batching method for its default type, then it will be used. A separate study could potentially be made where the difference between separate draw calls and a grouping to a vertex array or similar data structure with a single draw call would be.

3.5.1 Allegro

Two ALLEGRO_BITMAPs are created, one for the background and one for the image. They are created after the window is created, to make sure they are stored on VRAM and not regular RAM. While rendering the sprites to the screen, the bitmaps are held using al_hold_bitmap_drawing, batching the render calls which is the recommended technique when drawing several sprites from the same bitmap. Position and transformation are all saved in a separate struct.

3.5.2 SDL

The two SDL_Texture\_s are rendered to a SDL\_Renderer. SDL\_Texture is used to allow hardware acceleration. Position and transformation are all saved in a separate struct. One Object for the struct is created for each sprite to be displayed on the screen.

3.5.3 SFML

The background and the image are loaded into two separate sf\_\_Textures. Before each test, sf\_\_Sprites are created, equal to the number of tested sprites, which are all linked to the textures. Position and transformation are saved in the sprite.
3.6 Tested Computers

This section describes the general overview of the relevant hardware and software of the tested computers.

3.6.1 Desktop Computers

All of the tested desktop computers had a GPU separate from the CPU. The amount of VRAM available ranged from 1 GB to 8 GB, and all computers used DirectX 12 (the latest version of DirectX available at the time of writing). Two of the three desktop computers used OpenGL 4.6 (the latest version of OpenGL at the time of writing) and had 16 GB of RAM, while one used OpenGL 4.5 and had 8 GB of RAM. For a detailed list of information for each desktop computer, see Appendix A.

3.6.2 Laptop Computers

Unlike the desktop computers, none of the laptops had a dedicated GPU. Instead, the processor had an integrated GPU that is used for hardware acceleration. No dedicated graphics card also means that VRAM is borrowed from the regular system memory (RAM), which was a total of 8 GB for each laptop. All laptops used version 4.4 of OpenGL and DirectX 12 (the latest version of DirectX available at the time of writing). Three different laptops were tested. For a detailed list of information for each laptop computer, see Appendix A.
Chapter 4

Results

This chapter begins with some general testing information, before the results for each of the four types of test are displayed in diagrams, one per type of computer and test.

4.1 General Test Information

The testing took around 15 minutes per computer and graphical library. The average number of frames per library and type of computer is displayed below in 8 diagrams. Error bars are drawn on each data point representing the standard deviation of that result. Note that the error bars may be so small they are barely visible.
4.2 Static Sprites

In the static sprite test, the images rendered were rendered directly as is, and did not move during the testing. With one exception, the tested libraries follow a similarly-shaped asymptotic curve from few to many sprites rendered.

4.2.1 Desktop Computer

In figure 4.1 we see that the graphical performance of SDL and SFML were similar during all tests. SDL had a slightly better performance overall, increasing somewhat towards later tests. Allegro on the other hand had a fairly strong start in the first test, but then fell more and more behind with larger loads.

![Graph showing performance of Allegro, SDL, and SFML on desktop computer](image-url)
4.2.2 Laptop

As we can see in figure 4.2, Allegro is still falling behind a little in performance during the earlier tests on lower-end machines, but with larger sprite pools it actually exceeds both SDL and SFML. SFML was the strongest up to around 2 000 sprites, where it fell in with SDL up to 10 000. Note that all libraries had an increase in performance with 100 compared to 50 sprites.

Figure 4.2: Static sprites on laptop computer
4.3 Alpha Blended Sprites

In the alpha blend test, sprites were rendered statically but with varying levels of opacity. This means the program has to interpolate colors between the rendered image and the previously rendered color at the same location.

4.3.1 Desktop Computer

In figure 4.3 we can see the tests performed on alpha blended sprites on desktop computers. The results are very similar to the ones in figure 4.1 with the exception that SFML had less of a performance loss relative to SDL, and all three libraries had slightly weaker performances overall (around 2% down).

![Figure 4.3: Alpha-blended sprites on desktop computer](image-url)
4.3.2 Laptop

A drop in performance is seen when comparing alpha blended sprites in figure 4.4 to non-alpha blended sprites in figure 4.2. The general curve for the laptops is fairly similar, but while SDL and SFML lost on average 10% of their frames, Allegro lost only 5%.

![Graph showing performance comparison between Allegro, SDL, and SFML](image_url)

Figure 4.4: Alpha-blended sprites on laptop computer
4.4 Rotating Sprites

In the rotating sprite test, the images are rotated 360 degrees per second, calculated and rendered each frame.

4.4.1 Desktop Computer

As seen in figure 4.5, the general curves are fairly similar in shape to the static sprite tests. Allegro took a minor performance hit of around 2%, but SDL lost 11% (particularly towards the larger test sizes), and SFML 16% (similarly increasingly much towards larger tests). However, both SDL and SFML still outclass Allegro.

Figure 4.5: Rotating sprites on desktop computer
4.4.2 Laptop

Once again similar to the original static sprite results, figure 4.6 shows the same 2000-sprite breakpoint. While Allegro drops around 1% of its frames compared to static sprites on laptops, and SDL dropped 9%, SFML gained 11% more frames compared to the static tests, mainly through a large increase in the early tests.

Figure 4.6: Rotating sprites on laptop computer
4.5 Moving Sprites

In the moving sprites test, the images were moving at a fixed rate along one or both axes.

4.5.1 Desktop Computer

When it comes to moving sprites, SFML had a harder time keeping up with SDL and fell behind with a 10% performance loss in the later tests, as seen in figure 4.7. Allegro only dropped by around 2%, while SDL was entirely unaffected and precisely matched its static sprite results.

![Figure 4.7: Moving sprites on desktop computer](Image)
4.5.2 Laptop

In figure 4.8, we can see that the performance difference looks nearly identical to figure 4.2. Allegro and SFML render around 1% more frames than in the test of static sprites on laptops, while SDL rendered around 3% more frames, particularly in the earliest tests. Note how it matches SFML on the 50-sprite test.

![Graph showing Frames per second vs Sprites rendered for Allegro, SDL, and SFML](image-url)

Figure 4.8: Moving sprites on laptop computer
4.6 Result Summary

Separating results from both desktop and laptop computers into those of 'normal load' (up to 1000 sprites) and 'high load' (2000 to 10000 sprites) gives us four general areas of results. The average number of frames and standard deviation for each library is displayed in histograms in each category.

4.6.1 Desktop Computers

On desktop computers, the average number of frames produced by SDL and SFML was nearly equal, as can be seen in figure 4.9. SFML was slightly better for static and alpha-blended sprites, while SDL had an edge in rotating and moving sprites. Allegro however reached only 82% of the others’ performance, while also having the highest standard deviation (i.e. the least reliable result) which resulted in a coefficient of variation (CV) of 1.72%. SFML also had a fairly high standard deviation, but due to the higher average it resulted in a CV of 1.06%, or 29% lower than Allegro. SDL was by far the most stable, with a CV of 0.19%.

![Figure 4.9: Normal Load - Desktop Computer](image-url)
At higher loads, SFML began falling behind SDL, with an average of 16% less frames produced, as can be seen in figure 4.10. In the first three tests, SFML was just below 90% performance compared to SDL, while in the moving sprites test SFML only reached 70% of SDL’s performance. Allegro by this point had no more than half as many frames as SDL or SFML, and its CV had increased to 6.6%. SFML’s CV had likewise increased to 1.5%, while SDL experienced the greatest increase, to 1.6%.

![Figure 4.10: High Load - Desktop Computer](image)

### 4.6.2 Laptop Computers

For normal loads on laptops, Allegro and SDL were fairly equal, both around 20% behind SFML (see figure 4.11). SFML’s advantage compared to Allegro gradually diminished with greater numbers of sprites, while it stayed more clearly ahead compared to SDL. While SFML’s coefficient of variation was fairly high at 3.2%, most of that was due to three very high standard deviation values produced by only one of the three laptops (around 20-100 times higher than the other two laptops). If those values are corrected downwards to match that laptops other values compared to the first two laptops, the CV would be only 2.6%. Allegro and SDL had CVs 2.5% and 2.8% respectively. This is a fair bit higher than their desktop result equivalents, especially for SDL.
In the high-load laptop test (see figure 4.12), Allegro came out on top for the first time. None of the three libraries even managed to keep over the baseline acceptable 30 frames per second in the end, though Allegro came somewhat close. SFML and SDL performed very similarly, but while SFML had a decent CV of 2.4%, SDL had a CV of 5.8%, which was SDL’s highest CV by more than double.
Chapter 5

Discussion

Comparing all tests Allegro seems to be the weakest contender of the three libraries. The only point that Allegro performs the best is at over 1 000 sprites on laptops, and even then it was no more than 33% more frames at best. Comparing this to some of the tests on desktop computers where SDL was 180% faster than Allegro, it doesn’t seem that impressive. Unless you are aiming for very high loads specifically targeted to low-power machines, the real competition is between SDL and SFML.

5.1 Differences in Performance

As SFML seems to have the most balanced performance in total, SDL and Allegro will be compared to its results here. In figure 5.1, each of SDL’s and Allegro’s laptop results have been divided with SFML’s laptop results, and their desktop results with SFML’s desktop results. Thus, the graph displays how many percent of SFML’s performance the two other libraries produce for each number of sprites and on laptops or desktop computers, as well as the average between the two.

As we know from before, SDL is the strongest at desktop computers, and after being just around tied with SFML up to 500 sprites, it then incrementally reaches just over 20% higher performance. Similarly, Allegro performs the best on laptops rendering 1400 or more sprites, also reaching around 20% better than its SFML equivalent. Allegro greatly suffers on desktops, leaving its average at around 80% of SFML’s. SDL on the
other hand actually exceeds SFML’s average performance at around 1400 sprites or greater. Comparing the total of the entire test, it’s still slightly behind.

It seems that SDL is better able to take advantage of stronger hardware, especially when handling large sprite loads. On the flipside, Allegro performs well with weaker hardware and large loads. Likely, differences in the implementation of the rendering code is what is causing this disparity in result for different kinds of situations.

When comparing to the previous research that compares SDL and SFML from 2013, the performance difference is slightly higher. Looking at figure 5.1 for the 2000 sprites comparison, SDL on average managed to outperform SFML by about 7%, which is an increase of about 6 percentage points. It is important to note the point where SDL’s performance surpasses the SFML’s is somewhere between 1 000 and 2 000 sprites. Up to about 500 sprites, the difference between SDL and SFML does not change much.

Figure 5.1: Moving sprites on desktop computer
5.2 Reasons for the Performance Difference

Allegro, SDL, and SFML all use very similar methods to store their image data (textures) - all save images as arrays of bytes representing color channels of individual pixels. This in itself is unlikely to influence which program is the fastest. How the arrays are stored could be a difference, but all three of the libraries save their textures to video memory, meaning the impact of such storage differences would be quite small as well.

Each render call adds an significant overhead that needs to be processed. One common optimization in graphical application is to batch the rendering calls (waiting for several images and drawing them all at once), which significantly decreases the overhead. Allegro has built in render batching, which is used during the tests. SDL and SFML has no built in render batching for their default types. A possible explanation to why Allegro has the best performance for laptops when rendering a high number of sprites could be that the render call is the bottleneck for the application at that point. Removing rendering batching from Allegro decreased its performance by up to 90% for the laptops, changing the results completely.

The rendering back end for the multimedia libraries also differs, where Allegro and SDL both use Direct3D, while SFML uses OpenGL. The expected result would be that Direct3D applications would perform better on Windows, since Microsoft optimizes Direct3D for Windows only, while OpenGL is a cross-platform API and relies on the hardware manufactures to optimize the API implementation. Another advantage Direct3D holds is the more flexible memory handling, though this aspect is probably limited in the case for these multimedia libraries since only general purpose sprites were used. That in turn limits the rendering speed boost that could be gained when the memory handling can not be optimized for a specific purpose, instead it is applied to a general one.

Looking at the results, Direct3D does not seem to give a noticeable increase in performance, rather it’s almost the opposite. SFML, the only multimedia library that uses OpenGL, ended up being the library with the best overall performance. In the complete average, SFML is around 5% faster than SDL. SDL did however surpass SFML at some point between 1 000 and 2 000 sprites, going from being around 10% behind to 10% ahead. The conclusion is that the rendering back end for two-
dimensional graphical applications is not an important factor to consider when comparing multimedia libraries.

5.3 Potential Sources of Error

One big anomaly in the tests is that all multimedia libraries gains a performance boost for 100 sprites compared to 50 sprites when it comes to laptops. The biggest difference between desktop computers and laptops is that laptops uses an integrated GPU, using the normal system memory as VRAM. Integrated GPUs are generally weaker than dedicated GPUs, and require less power to run.

Being weaker does not explain why it gains a performance boost when put under heavier load. All tests are performed 20 times, but not directly after each other. Instead, all tests are run to completion, then the whole process is repeated again, a total of 20 times. That makes startup time an unlikely factor, especially since the deviation from the norm for the first test is only a percentage or two higher than the other tests, nothing that explains the big difference.

One possible explanation for the performance peaking on the second test is the power plans on Windows 10. The default power plan is balanced, which increases the CPU speed when under heavy load, and decreases it when under low load \[32\]. It is thus possible that Windows uses a slightly lower clock speed on the tests with 50 sprites, while boosting up to a higher clock speed when reaching 100 or more sprites.

The Windows 10 power plan also depends on whether the computer’s power cord was plugged in or not. The battery state of the laptops was not a controlled factor during the tests, making it a possible source of error. The difference between the first and the second test is about 10-20%. Thus it is possible that plugging in the laptop could increase the performance for some or all of the tests on laptops.

Another possible influence to the results is the group’s relative inexperience in the optimization of the tested graphics libraries. All of the three multimedia libraries had in some degree lacking documentation and information. While SFML has a rather comprehensive wiki, Allegro, and even more so SDL, sometimes have near nonexistent information on their functionality and design. At times you are simply met with a
one-sentence page and a large box saying "add your code example here". This can make it difficult to gather information on how to most efficiently draw in the program. To minimize this problem, no advanced functions were used, instead only the standard texture structures and rendering functions were used.

To note is that we used the uncorrected sample standard deviation formula for standard deviations, where it could have possibly been advised to use corrected sample standard deviation due to the small sample size (see figure 5.2). This seems to lower all standard deviations by around 2.5%, which is not a very large amount.

The total number of tested computers were on the lower side, consisting of three desktops and three laptops. The results could therefore not claim to represent the rendering performance for laptops and desktops as a whole, instead it should be used for a guide on how the multimedia libraries perform on different hardware. The three laptops can be assumed to at least somewhat adequately represent the normal hardware used for office laptops, as they are on the average to weak side of the most common laptop types available in northern Europe at this time.

\[
\sqrt{\frac{1}{N} \sum (x - \bar{x})^2} \quad \sqrt{\frac{1}{N-1} \sum (x - \bar{x})^2}
\]

Figure 5.2: Uncorrected (left) and corrected (right) standard deviation
Chapter 6

Conclusion

Each of the three multimedia libraries ended up with a preferred situation in regards to hardware power and number of images. However, their total results are in no way equal. First off, in the unlikely situation that you are rendering very high numbers of sprites specifically on low-end machines, Allegro is a good choice. In any other situation, including laptops under low strain, it is far behind both SDL and SFML. Especially its result on desktop computers fall behind, reaching less than 50% of SFML’s performance at 10 000 sprites (and less than 40% of SDL’s performance at the same point).

If you on the other hand know you are going to render a mass of sprites on powerful hardware, SDL is the clear winner. It can efficiently use a better computer to handle large amounts of images relatively effectively. In fact, it is also equal to SFML for lower load on desktop computers, meaning that if you are exclusively developing for desktops, it might be the best option. Its laptop results are worse, but start to catch up to SFML around 2 000 sprites, leaving it less behind.

For a balanced rendering system aimed for both laptops and desktop computers, SFML is probably the best choice, due to it having the best overall performance. It has the highest average performance up to just over 1 000 sprites, and does not fall very far behind SDL even after that point. Even at 10 000 sprites, SFML outperforms SDL on laptops. If you expect to use your rendering system on both low and high power machines, and not always at 2000+ sprite loads, this means that SFML is prime candidate of the three tested multimedia libraries.
6.1 Future Research

There are several potentially relevant research areas on the differences of rendering systems to explore in the future. This study was fairly small in scope, and as all of the tested libraries update continually comparisons may quickly become obsolete as one or another library improves past the others. Below is a list of some relevant further questions that have been considered:

- Comparing the multimedia libraries on multiple operating systems.
- Comparing the default rendering option (sprite/bitmap) to a vertex array or similar data structure, which would significantly decrease rendering overhead on all multimedia libraries for large loads.
- Only six different computers were tested, three desktops and three laptops. Running the tests on a wider range of computers, especially ones with different levels of hardware, would yield interesting results since the multimedia libraries are so clearly and differently affected by different hardware.
- The multimedia libraries experienced a difference of up to 25% when comparing different graphical operations, for example static sprites compared to rotating ones. A total of four different graphical operations were tested in this report, something that would be interesting to see increased. Another common graphical operation is the use of shaders to modify the rendered images.
- All three multimedia libraries build on either OpenGL or Direct3D. One interesting comparison would thus be to create a reference program purely in these two graphical APIs, that the other multimedia libraries could be compared to.
- It would also be interesting to increase the number of multimedia libraries that are tested. Two potential candidates are Cinder and OpenFrameworks.
Bibliography


[22] Window module (SFML / Learn / 2.5.1 Documentation). URL: https://www.sfml-dev.org/documentation/2.5.1/group__window.php (accessed: 22.03.2019).


Appendix A

Tested Computers

Below is the general graphics-related specifications of the six tested computers, in order of descending approximate power. Their individual results can be found on github: [https://github.com/antonlc/multimedia-library-comparison](https://github.com/antonlc/multimedia-library-comparison).

### A.1 Computer 1 (Desktop)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Core i7-6700k (4.0 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>16 GB</td>
</tr>
<tr>
<td>VRAM</td>
<td>8 GB (GDDR5X)</td>
</tr>
<tr>
<td>Storage</td>
<td>1 TB SSD (NVMe)</td>
</tr>
<tr>
<td>GPU</td>
<td>NVidia GeForce GTX 1080</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Version 4.6</td>
</tr>
<tr>
<td>DirectX</td>
<td>Version 12</td>
</tr>
</tbody>
</table>
### A.2 Computer 2 (Desktop)

<table>
<thead>
<tr>
<th><strong>Processor</strong></th>
<th>Intel Core i5-6400k (2.7 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM</strong></td>
<td>16 GB</td>
</tr>
<tr>
<td><strong>VRAM</strong></td>
<td>6 GB (GDDR5)</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>256 GB SSD</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>NVidia GeForce GTX 1060</td>
</tr>
<tr>
<td><strong>OpenGL</strong></td>
<td>Version 4.6</td>
</tr>
<tr>
<td><strong>DirectX</strong></td>
<td>Version 12</td>
</tr>
</tbody>
</table>

### A.3 Computer 3 (Desktop)

<table>
<thead>
<tr>
<th><strong>Processor</strong></th>
<th>Intel Core i5-2320k (3.0 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM</strong></td>
<td>8 GB</td>
</tr>
<tr>
<td><strong>VRAM</strong></td>
<td>1 GB (GDDR5)</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>1 TB HDD</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>AMD Radeon HD 7700</td>
</tr>
<tr>
<td><strong>OpenGL</strong></td>
<td>Version 4.5</td>
</tr>
<tr>
<td><strong>DirectX</strong></td>
<td>Version 12</td>
</tr>
</tbody>
</table>

### A.4 Computer 4 (Laptop)

<table>
<thead>
<tr>
<th><strong>Processor</strong></th>
<th>Intel Core i5-6200U (2.3 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM</strong></td>
<td>8 GB</td>
</tr>
<tr>
<td><strong>VRAM</strong></td>
<td>(none)</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>256 GB SSD</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>Intel HD Graphics 520</td>
</tr>
<tr>
<td><strong>OpenGL</strong></td>
<td>Version 4.4</td>
</tr>
<tr>
<td><strong>DirectX</strong></td>
<td>Version 12</td>
</tr>
</tbody>
</table>
## A.5 Computer 5 (Laptop)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Core i5-6200U (2.3 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>8 GB</td>
</tr>
<tr>
<td>VRAM</td>
<td>(none)</td>
</tr>
<tr>
<td>Storage</td>
<td>128 GB SSD</td>
</tr>
<tr>
<td>GPU</td>
<td>Intel HD Graphics 520</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Version 4.4</td>
</tr>
<tr>
<td>DirectX</td>
<td>Version 12</td>
</tr>
</tbody>
</table>

## A.6 Computer 6 (Laptop)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Core m3-7Y30U (1.0 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>8 GB</td>
</tr>
<tr>
<td>VRAM</td>
<td>(none)</td>
</tr>
<tr>
<td>Storage</td>
<td>256 GB SSD</td>
</tr>
<tr>
<td>GPU</td>
<td>Intel HD Graphics 615</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Version 4.4</td>
</tr>
<tr>
<td>DirectX</td>
<td>Version 12</td>
</tr>
</tbody>
</table>