**Parallel Versions of SVG to ASS Transformer Implemented by Pthreads and CUDA**

**Teng Yue 1007826792**

**Wenrui Xu 1008313228**

**Jiaming Xu 1007698831**

Dec 8, 2021

# ABSTRACT

Nowadays, with the development of the self-media industry, many video processing and production positions have emerged. A very common scenario is: video workers post-process the video, adding graphics, subtitles and other elements. In particular, for adding graphic elements, video workers often use SVG files to add to the video to achieve more gorgeous artistic effects. In order to add a specific SVG file to the video, it usually needs to be converted to an ASS file. The ASS format is a commonly used subtitle file format. However, if a single-threaded program is used to convert an SVG file into an ASS file, it takes a lot of time.

In order to speed up the conversion, we will rewrite the single-threaded conversion program into a multi-threaded program, and process the conversion operations of each layer of the SVG file in parallel. The parallel conversion programs can shorten the conversion time and improve the work efficiency of video workers.

# CONTENTS

[ABSTRACT 1](#_Toc89791473)

[Chapter 1 Introduction 4](#_Toc89791475)

[1.1 Backgrounds and Aims 4](#_Toc89791476)

[1.2 Report Structure 6](#_Toc89791477)

[Chapter 2 SVG and ASS 7](#_Toc89791478)

[2.1 SVG 7](#_Toc89791479)

[2.2 ASS 8](#_Toc89791480)

[Chapter 3 Pthreads and CUDA 10](#_Toc89791481)

[3.1 Pthreads 10](#_Toc89791482)

[3.2 CUDA 12](#_Toc89791483)

[Chapter 4 Implementation of Parallel Versions of SVG to ASS Transformer 13](#_Toc89791484)

[4.1 Basic Implementation 13](#_Toc89791485)

[4.2 Implementation by Pthreads 13](#_Toc89791486)

[4.3 Implementation by CUDA 13](#_Toc89791487)

[Chapter 5 Performance Evaluation 14](#_Toc89791488)

[5.1 Performance Evaluation of the Pthreads Version 14](#_Toc89791489)

[5.2 Performance Evaluation of the CUDA Version 14](#_Toc89791490)

[Chapter 6 Conclusion and Further Work 15](#_Toc89791491)

[6.1 Conclusion 15](#_Toc89791492)

[6.2 Further Work 15](#_Toc89791493)

[Reference 16](#_Toc89791494)

## Chapter 1 Introduction

### 1.1 Backgrounds and Aims

Nowadays, the number of mobile devices continues to increase. Compared with traditional TV entertainment methods, entertainment on mobile devices, including YouTube, TikTok, etc., is more and more popular and favored by people. With the outbreak of COVID-19, more people choose to stay at home to work, and the time spent at home has increased significantly. Therefore, people's demand for fast entertainment has increased greatly. In this context, the demand for self-media services such as short video creation has increased significantly, and more and more people have joined the creation of streaming media.

For video workers, they usually record the video first, and then use video editing software to add the desired elements to the video, including animation, subtitles, etc. For adding graphic elements, video workers often use SVG files to add to the video to achieve more gorgeous artistic effects.

Scalable Vector Graphics (SVG) are an XML-based markup language for describing two-dimensional based vector graphics.

As such, it's a text-based, open Web standard for describing images that can be rendered cleanly at any size and are designed specifically to work well with other web standards including CSS, DOM, JavaScript, and SMIL. SVG is, essentially, to graphics what HTML is to text.

SVG images and their related behaviors are defined in XML text files, which means they can be searched, indexed, scripted, and compressed. Additionally, this means they can be created and edited with any text editor or with drawing software.

Compared to classic bitmapped image formats such as JPEG or PNG, SVG-format vector images can be rendered at any size without loss of quality and can be easily localized by updating the text within them, without the need of a graphical editor to do so. With proper libraries, SVG files can even be localized on-the-fly.

SVG has been developed by the World Wide Web Consortium (W3C) since 1999.

In order to add a specific SVG file to the video, it usually needs to be converted to an ASS file. Then, ASS files can be imported into the video to achieve artistic effects.

SubStation Alpha (or Sub Station Alpha), abbreviated SSA, is a subtitle file format created by CS Low (also known as Kotus) that allows for more advanced subtitles than the conventional SRT and similar formats. It is also the name of the popular, now discontinued tool used to edit subtitles.

This subtitle format is frequently used in anime fansubs, either to overlay subtitles onto video while it is being encoded (hardsubbing), or to store subtitle data alongside video data, often in a Matroska (MKV) container (softsubbing). It's not commonly used professionally except for Crunchyroll.

The current version of SSA is v4.08. There are many freeware and open-source subtitling applications that support the SSA format.

Advanced SubStation Alpha (ASS) is a script for more advanced subtitles than SSA. It is technically SSA v4+. It is able to produce anything from simple texts to manual graphic editing used in karaoke. There are few programs designed to create these scripts. The main feature of ASS is it has more specifications than normal SSA, like in styles programming.

However, in the existing commonly used SVG to ASS conversion software, the conversion speed is very slow. This is because there are so many layers in the SVG file. There may be tens of thousands or even 100,000 layers in an SVG file, and traditional conversion software needs to process the layers layer by layer and convert them to the corresponding ASS file code, so the entire conversion process will take a lot of time. In order to speed up the conversion of an SVG file to an ASS file, save time, and improve the work efficiency of video workers, we will use Pthreads and CUDA to rewrite the single-threaded conversion program and improve it into parallel programs. We will evaluate and analyze the execution efficiency of parallel programs and compare them with the single-threaded program.

### 1.2 Report Structure

Chapter 1, “Introduction”, presents the background of this project and our objective.

Chapter 2, “SVG and ASS”, provides some basic information about the SVG and ASS.

Chapter 3, “Pthread and CUDA”, presents the technologies of Pthread and CUDA, which are used in this project.

Chapter 4, “Implementation of Parallel Versions of SVG to ASS Transformer”, discusses how we use Pthread and CUDA to implement the parallel versions of the transformer.

Chapter 5, “Performance Evaluation”, presents the analysis of both the Pthread version and the CUDA version, and compare their performance.

Chapter 6, “Conclusion and Further Work”, concludes our work with a summary of the outcomes, as well as our future works.

## Chapter 2 SVG and ASS

### 2.1 SVG

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. SVG images can thus be scaled in size without loss of quality, and SVG files can be searched, indexed, scripted, and compressed. The XML text files can be created and edited with text editors or vector graphics editors, and are rendered by the most-used web browsers.

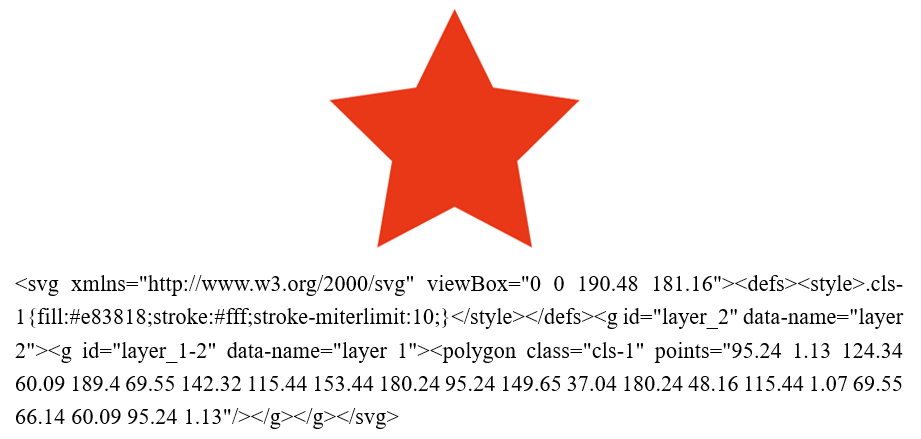


Figure 2.1 A SVG file and its output graph

SVG documents are built upon a regular XML document tree, consisting primarily of a header, processing instructions, comments, XML elements and attributes. It allows the creator to conveniently embed arbitrary information inside of the file. SVG is well suited for graphics rich environments. SVG can be used for GIS, embedded systems, location-based services, animated picture messaging, multimedia messaging, animation and interactive graphics, entertainment, e-Business, and graphic user interfaces. After integrating 3D into SVG, it can even play an important role in many fields such as product demonstration, city planning, site exhibition, 3D e-Business (such as 3D shopping malls), etc.

SVG uses a “painter model” for rendering. Paint is applied in successive operations to the output device such that each operation paints over some area of the output device. When the area overlaps a previously painted area, the new paint partially or completely obscures the old. When the paint is not completely opaque, the result on the output device is defined by the mathematical rules for composing. Elements in an SVG document fragment have an implicit drawing order. Elements that appear first in the document tree are rendered first; subsequent elements are drawn on top of the previous elements, taking into account opacity, blending, filters, clipping and masking.

The figure 2.1 shows a typical SVG file and its output graph.

### 2.2 ASS

SubStation Alpha (or Sub Station Alpha), abbreviated SSA, is a subtitle file format that allows for more advanced subtitles than the conventional SRT and similar formats. It is also the name of the popular, now discontinued tool used to edit subtitles.

This subtitle format is frequently used in anime fansubs, either to overlay subtitles onto video while it is being encoded (hardsubbing), or to store subtitle data alongside video data, often in a Matroska (MKV) container (softsubbing).



Figure 2.2 A ASS file and its output graph

Advanced SubStation Alpha (ASS) is a script for more advanced subtitles than SSA. It is technically SSA v4+. It is able to produce anything from simple texts to manual graphic editing. There are few programs designed to create these scripts. The main feature of ASS is it has more specifications than normal SSA, like in styles programming.

The ASS file format allows for the formatting and stylizing of subtitle text and is very popular in Anime and Karaoke projects.

The figure 2.2 shows a typical ASS file and its output graph.

## 

## Chapter 3 Pthreads and CUDA

### 3.1 Pthreads

POSIX Threads, commonly known as Pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API.

The POSIX thread libraries are a standards-based thread API for C/C++. It allows one to spawn a new concurrent process flow. It is most effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing. Threads require less overhead than "forking" or spawning a new process because the system does not initialize a new system virtual memory space and environment for the process. While most effective on a multiprocessor system, gains are also found on uniprocessor systems which exploit latency in I/O and other system functions which may halt process execution. (One thread may execute while another is waiting for I/O or some other system latency.) Parallel programming technologies such as MPI and PVM are used in a distributed computing environment while threads are limited to a single computer system. All threads within a process share the same address space. A thread is spawned by defining a function and it's arguments which will be processed in the thread. The purpose of using the POSIX thread library in your software is to execute software faster.

There are around 100 threads procedures, all prefixed pthread\_ and they can be categorized into four groups:

* Thread management - creating, joining threads etc.
* Mutexes
* Condition variables
* Synchronization between threads using read/write locks and barriers.

Thread operations include thread creation, termination, synchronization (joins, blocking), scheduling, data management and process interaction.

Mutexes are used to prevent data inconsistencies due to race conditions. A race condition often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depend on the order in which these operations are performed. Mutexes are used for serializing shared resources. Anytime a global resource is accessed by more than one thread the resource should have a Mutex associated with it. One can apply a mutex to protect a segment of memory ("critical region") from other threads. Mutexes can be applied only to threads in a single process and do not work between processes as do semaphores.

A join is performed when one wants to wait for a thread to finish. A thread calling routine may launch multiple threads then wait for them to finish to get the results. One wait for the completion of the threads with a join.

A condition variable is a variable of type pthread\_cond\_t and is used with the appropriate functions for waiting and later, process continuation. The condition variable mechanism allows threads to suspend execution and relinquish the processor until some condition is true. A condition variable must always be associated with a mutex to avoid a race condition created by one thread preparing to wait and another thread which may signal the condition before the first thread actually waits on it resulting in a deadlock. The thread will be perpetually waiting for a signal that is never sent. Any mutex can be used, there is no explicit link between the mutex and the condition variable.

When this option is enabled, each thread may have its own scheduling properties. Scheduling attributes may be specified:

* during thread creation
* by dynamically by changing the attributes of a thread already created
* by defining the effect of a mutex on the thread's scheduling when creating a mutex
* by dynamically changing the scheduling of a thread during synchronization operations.

The threads library provides default values that are sufficient for most cases.

### 3.2 CUDA

CUDA (or Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing unit (GPU) for general purpose processing – an approach called general-purpose computing on GPUs (GPGPU). CUDA is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The main difference between a CPU and a GPU is that a CPU is a serial processor while the GPU is a stream processor. A serial processor, based on the Von Neumann architecture executes instructions sequentially. Each instruction is fetched and executed by the CPU one at a time. A stream processor on the other hand executes a function (kernel) on a set of input data (stream) simultaneously. The input elements are passed into the kernel and processed independently without dependencies among other elements. This allows the program to be executed in a parallel fashion.

CUDA is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. CUDA-powered GPUs also support programming frameworks such as OpenMP, OpenACC and OpenCL.

## Chapter 4 Implementation of Parallel Versions of SVG to ASS Transformer

### 4.1 Basic Implementation

### 4.2 Implementation by Pthreads

### 4.3 Implementation by CUDA

We implemented this module by copying the data used by pthreads into GPU with CUDA library. First, we use cudaMalloc to malloc space for the data passed in. After that, we use cudaMemcpy to copy the local variable into GPU storage and create the threads in GPUs and run the functions. After that, we will copy the GPU storage back into local storage.

## Chapter 5 Performance Evaluation

### 5.1 Performance Evaluation of the Pthreads Version

### 5.2 Performance Evaluation of the CUDA Version

## Chapter 6 Conclusion and Further Work

### 6.1 Conclusion

### 6.2 Further Work

First, we can try to reconstruct the code to modularize it. After that, we can use pipeline to accelerate the multi-thread process instead of using the pthread library.

Second, we can modify the parser module to apply this program into more file layouts.

Last but not least, we can make this program become an interface for others to use it more easily.

## Reference

Robey, R., Zamora, Y. (2021) *Parallel and High Performance Computing*. Greenwich. USA: Manning Publications.

Cheng, J., Grossman, M., McKercher, T., Chapman, B. (2014) *Professional CUDA® C programming.* Indianapolis. USA: Wrox.

Rauber, T., Rünger, G. (2013) *Parallel Programming for Multicore and Cluster Systems.* Berlin, German: Springer

Czarnul, P. (2018) *Parallel Programming for Modern High Performance Computing Systems.* Boca Raton, USA: Chapman and Hall/CRC.