ECE1747H Parallel Programming Seminar Project Report

SVG-to-ASS File Format Converter: A multithreading approach

Jiaming Xu[1], Teng Yue[2], and Wenrui Xu[1]

[1]1007698831, Department of Electrical & Computer Engineering, University of Toronto
[2]1007826792, Department of Electrical & Computer Engineering, University of Toronto
[3]1008313228, Department of Electrical & Computer Engineering, University of Toronto

December 17, 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 Scalable Vector Graphics (SVG) files to add to the video as vector paths to achieve more gorgeous artistic effects, including filtering, motion graphic (MG) animations and etc. In order to add a specific SVG file to the video, other than inserting it directly into video editing software, an alternative way to achieve that is to convert it to an Advanced SubStation Alpha (ASS) file. With the requirement for making subtitles, this method is more convenient and efficiency.
ASS format is a commonly used subtitle file format with advanced support for graphs. However, using a single-threaded program to do conversion is time-costing. Thus, we describe an approach rewriting the single-threaded conversion program into a multi-threaded program, and processing 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.

Keywords CUDA, Pthread, SVG, Subtitles

1 Introduction

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.

Subtitling a range of topic in media producing, and recently people would love to create graphs with the subtitles to make animation and other effects. However, graph creating in subtitles can't directly use universal picture or graph formats, such as JPEG, SVG or BMP, instead, it needs a drawing process, which is encoded by a few of special scheme. Creating those drawing graph has always taken a lot of time for the effect subtitle producer to make it up. So making those drawing scheme by converting from some universal format is essential.
This paper is a report for a paralleled program approach. We focused on programming a converter from SVG format to a specific drawing scheme under ASS format. There are some features of the two vector graph format or scheme to tell.

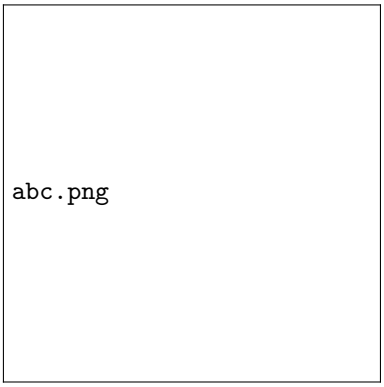


Figure 1.1: A sample figure of a SVG file

For instance, regular graph of star in SVG format will be encoded as follows:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0
    190.48 181.16"><defs></defs><g id="Layer_2" data
    -name="Layer 2"><g id="Layer_1-2" data-name="
    Layer 1"><polygon style="fill:#e83818;stroke:#fff;
    stroke-miterlimit:10;" points="95.24 1.13 124.34 60.09
    189.4 69.55 142.32 115.44 153.44 180.24 95.24 149.65
    37.04 180.24 48.16 115.44 1.07 69.55 66.14 60.09 95.24
    1.13"/></g></g></svg>
```

It's easy to tell this is a XML-based format. The root tag indicates the format of SVG, with a viewBox to limit the border of the graph. Tag <defs> is a definition part which could be able to define particular classes to apply styles in groups, the example stayed in empty, which we'll discuss deeper in the pre-processing section. Tag <g> refers to "group", which indicates the grouping attribute and relationships of those layers. In this simple example, the only drawing section is the <polygon> tag, which defines the points(vertices) and the style of the polygon layer.

On the other hand, if we would like to create a same graph in ASS drawing scheme, the encoded graph will be like:

```
Dialogue: 0,0:00:00.00,0:00:01.00,Default,Sign,0,0,0,,{\an7\1c
    &H1838E8&\1a&H00&\3c&HFF0F00&\3a&H00&\
    bord0\shad0\p1}m 95.2 1.1 l 124.3 60.1 l 189.4 69.5 l
    142.3 115.4 l 153.4 180.2 l 95.2 149.7 l 37 180.2 l 48.2
    115.4 l 1.1 69.5 l 66.1 60.1 l 95.2 1.1 {\p0}
```

This scheme could be save directly as a .ass file to create a Advanced SubAlpha Standard subtitle file.

2 Background

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.

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.

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).

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.

2.3 Pthread

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 readwrite 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.

2.4 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.
The procedure of using CUDA is using cudaMalloc first to allocate memory space in GPU. Then use cudaMemcpy to copy the local storage to GPU storage at first, and then create the tasks and pass in the parameters' pointers. After the execution, the CPU will copy the calculated results back from the GPU storage and use cudaFree to delete the GPU storage.

3 SVG2ASS

Our program is a scripting program interact with command line. It supports different options to construct the ASS file with custom settings, and also provides different ways for input and output. It could be deployed with a Web-based GUI, which provides an alternative way to do rapid test and verification.

3.1 preprocessing

In order to verify our converter outputs and test its performance, we prepared a bunch of sample graphs in SVG format. To achieve this, we firstly find some pictures as samples: A picture of Bahen Centre (Sample1), a logo graph of UofT (Sample2), and a flag graph of Canada (Flag), which are all in JPEG format. Secondly, we arrange to use Adobe Illustrator to convert those picture into graph consisting of paths and finally export them into SVG format in different presets to create different difficulties.

For instance, Sample 1 is a picture of Bahen Centre, the original picture is like:

And, we could be able to arrange vectorization with different presets: gray, 16-color, 6-color, 3-color, high quality and low quality. Different quality (presets) can result in different outcome of vectorization, and also different difficulty: the higher quality is, the more complex the path would be, to fit the requirement of outcome quality.

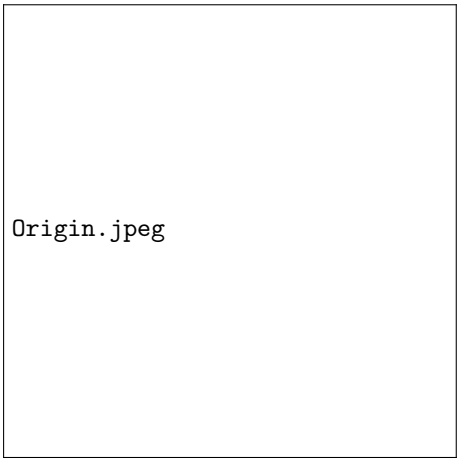


Figure 3.1: The original figure

This procedure is also included in a standard workflow of creating graph subtitles: we need to have the universal SVG files before our conversion.

3.2 Design

This section will introduce how the different parts work in our system.

3.2.1 nxml

Nxml is the main parser of SVG file, which is similar to decoder in function. The parser is adapted from the original repository which we didn't focus on, regarding it as a black-box function. What we majorly arranged, is the way it assigned tasks to the svg2ass controller. After it received the feeding parameters from the main function, it will record the status of the file parser from beginning, Then it will repeatedly goto the next tag by search for the "<". This is what we could make into parallelization.

3.2.2 svg2ass

Svg2ass is the major function controller of the system, which take the feed-in from the nxml by a struct and generate output in branch depending on the condition. It will call the ass_line to

3.3 Implementation

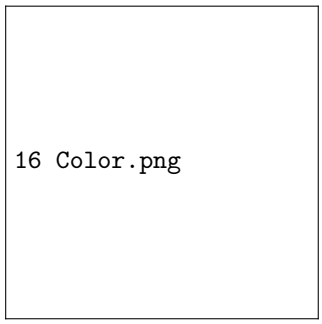
This section describes our system implementation of the paralleled svg2ass.

3.3.1 Sequential Implementation

The sequential implementation is simple, which we almost have no modification on it. The only thing we added is the preprocessing module to fix the color incompatible error. The main function takes the parameters from the user and process the transaction by branching different condition with a status code and call the nxml to start processing. The transactions of writing and calculating will be assigned in sequence along with the parser goes on,

(a) 3Color.svg

(b) 6Coloe.svg



(c) 16Color.svg

(d) Gray.svg

(e) HighQuality.svg

(f) LowQuality.svg

Figure 3.2: Samples of vectored figures.

until the nxml parser reaches an end, then it will return back to main function and see if there's multiple files to be processed, if nothing left, then the program will returned.

3.3.2 Pthread Implementation

The Pthread implementation changes the way to assign tasks. Firstly, we designed a finite status machine to record the current status allocated with parser. The advantage to do that is to monitor the current status of parser and communicate the information of queue to control the thread operations. Secondly, we repeatedly goto the next tag while sending tasks once the FSM reach the processing status. Finally, we free all the mutex and return all the thread created by the queue.
We arranged several structure and modification to manage multithreading.

3.3.3 CUDA Implementation

All the extra implementations from pthread implementation are in the nxml.c file. First, we get the pointers to store the node, user and event information in the GPU storage. In addition, we set up a pointer which is used to point to the result in GPU. Then, we malloc the memory space in GPU for every pointers. After that, we creates the tasks and pass in the parameters. Then we'll put it in the queue like pthread. At last, in every thread we created, when it's idle, it'll query the queue for more tasks which speed up the program.

4 Experiment

This section will introduce how we set up our experiment environment and how we arranged the experiment.

4.1 Environment

As our implementation applied with POSIX threading, the runtime environment have to be on Linux Platform. We did our experiment in the environment below:

- CPU: Intel i7-10700K, Assigned 16GB + 8GB Swap, Ubuntu 18.04.3.
- GPU: NVIDIA GeForce RTX 2060
- CUDA 11.5.1

4.2 Procedure

In order to evaluate the performance of different versions, we have found sixteen SVG files, which have different amount of tags. The range of length among these files is very large, so that we can evaluate the performance in terms of the input scale.
For each version, we test each file for ten times and compute the average running time. Finally, for each input file, we compare the running time of different implementation versions, and we compute the speed up.

4.3 Results

As the figure shows, in all our Pthreads versions, when the scale of the input file is not large enough, the average running time is larger than the sequential version. More specifically, this situation is true for all the figures that has less tag number than 2_HQ, which is 464. This is because the creation and destroy of threads may cost extra time, and we use locks to control critical section access, which also brings extra overhead. However, when the input file is large enough, the overhead of the thread creation and synchronization takes a very small part of the total running time. Therefore, the Pthread versions contribute to notable decrease of the running time, compared with the sequential version. Therefore, the performance of each Pthreads version is better than the sequential one.
For different Pthreads versions in terms of the number of threads, when the number raises, the running time of processing a certain file obviously decreases. Since greater

threads number means higher degree of parallel, more tasks can be done in a unit time slot. Therefore, in our implementation, a version with more threads has less running time. When we use 16 threads, it is up to 1.89 times speed-up, compared with the sequential version.
For our implementation of the CUDA version, the running time decrease greatly when the input file scale is relatively large, it is up to 6.94 times speed-up. When the input file scale is quite small, the running time of the CUDA version is larger than the sequential version, because the initialization cost is from the deep-copy from Core memory to GPU memory (cudaMalloc). As a result, the memory allocation will have a significant overhead in the small-scale condition.

5 Conclusion

Our work is on Github at https://github.com/Awesome-guys-in-ECE1747/Seminar_Project.

Related Work

This project is adapted on an open-source converter, svg2ass. https://github.com/irrwahn/svg2ass

Future 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.

References

[1] Aegisub Override Tags from Aegisub Official Documentation, https://aeg-dev.github.io/AegiSite/docs/3.2/ass_tags/

[2] Turtle Graphic in Python 3.10 Official Documentation, https://docs.python.org/3/library/turtle.html

[3] GUI repository by qgustavor, https://github.com/qgustavor/svg2ass-gui

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

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

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

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

Filename	Average Running Time	Valid Tag Quantity	Character Quantity
Test/Circle.svg	0.21	9	251
Test/Rect.svg	0.24	9	256
Test/Star.svg	0.20	9	346
Test/Flag.svg	0.69	12	2142
Sample2/Gray.svg	13.96	193	58006
Sample2/LQ.svg	13.85	193	58711
Sample2/16Color.svg	27.32	216	110856
Sample2/6Color.svg	25.96	216	110856
Sample2/3Color.svg	26.06	216	110856
Sample2/HQ.svg	24.04	464	93236
Sample1/3Color.svg	105.49	1141	397198
Sample1/LQ.svg	138.79	1555	603302
Sample1/6Color.svg	185.52	2349	973603
Sample1/Gray.svg	243.57	3385	939956
Sample1/16Color.svg	262.92	3960	973472
Sample1/HQ.svg	856.55	15007	2955127

Table 1: Sequential Program Running Statstics

Filename	16 Threads	8 Threads	4 Threads
Test/Circle.svg	1.53	1.54	1.49
Test/Rect.svg	1.62	1.64	1.93
Test/Star.svg	2.21	2.25	2.65
Test/Flag.svg	2.75	2.81	3.30
Sample2/Gray.svg	21.64	24.90	26.22
Sample2/LQ.svg	21.17	22.04	25.68
Sample2/16Color.svg	30.33	31.53	36.78
Sample2/6Color.svg	29.08	31.11	35.21
Sample2/3Color.svg	29.19	31.66	35.32
Sample2/HQ.svg	26.69	28.82	32.03
Sample1/3Color.svg	101.27	115.45	137.32
Sample1/LQ.svg	130.46	150.03	176.64
Sample1/6Color.svg	168.82	194.14	228.42
Sample1/Gray.svg	211.90	241.57	286.49
Sample1/16Color.svg	223.49	252.54	301.93
Sample1/HQ.svg	453.97	508.45	612.86

Table 2: Pthread Program Running Statstics

Filename	Sequential	Pthreads	CUDA
Test/Circle.svg	0.21	1.53	0.84
Test/Rect.svg	0.24	1.62	0.92
Test/Star.svg	0.20	2.21	1.12
Test/Flag.svg	0.69	2.75	1.37
Sample2/Gray.svg	13.96	21.64	6.57
Sample2/LQ.svg	13.85	21.17	6.12
Sample2/16Color.svg	27.32	30.33	10.15
Sample2/6Color.svg	25.96	29.08	10.73
Sample2/3Color.svg	26.06	29.19	11.45
Sample2/HQ.svg	24.04	26.69	12.21
Sample1/3Color.svg	105.49	101.27	72.35
Sample1/LQ.svg	138.79	130.46	78.91
Sample1/6Color.svg	185.52	168.82	77.52
Sample1/Gray.svg	243.57	211.90	77.34
Sample1/16Color.svg	262.92	223.49	79.16
Sample1/HQ.svg	856.55	453.97	123.42

Table 3: Pthread Program Running Statstics