The Rendering Optimization of SVG Player on Mobile Devices

Zhijie Qiu, Lei Luo, Jixian Zhang

1 School of Computer Science & Engineering, University of Electronic Science and Technology of China Chengdu, qzhijie@uestc.edu.cn
2 School of Computer Science & Engineering, University of Electronic Science and Technology of China Chengdu, lluo@uestc.edu.cn
3 School of Information Science and Engineering, Yunnan University, Kunming, denonji@163.com

Abstract

With the 3G development, more and more rich media applications appear on mobile devices. The efficiency of scalable vector graphics player is very crucial for constrained mobile devices. In this paper, we propose a partial rendering algorithm which dynamically traces the invalid areas of elements in the scene and rationally divides the scene into several partial rendering areas. Compared with the traditional algorithms, this algorithm can obviously improve performance on high resolution mobile devices. We also present a new contour algorithm for fitting circular arc, which uses midpoint rule to calculate the control points of cubic bezier curve. This algorithm has lower complexity than the traditional algorithms and is suitable for arc drawing on mobile devices. The test results show that above two algorithms have good effect on rendering optimization.

Keywords: Scalable Vector Graphics, Partial Rendering Algorithm, Arc Contour Algorithm

1. Introduction

As a 2D vector graphics description language for mobile devices, SVG Tiny 1.2 specification [1] has aroused increasing number of attention. On the one hand, many standards and schemes are based on SVG Tiny 1.2 specification, such as MPEG-4 LASeR [2], 3GPP DIMS [3], and Nokia MORE [4], etc. On the other hand, some browsers have begun to support playing SVG Tiny documents on mobile devices, such as Opera mobile, Android webkit, Nokia WRT and so on. But without exception, these technologies all need to use SVG Tiny engine. The efficiency of the engine is very crucial. The execution process of SVG Tiny engine mainly includes parsing, layout, animation, rendering, and other aspects. Rendering is the most time-consuming part and the optimization for it needs to be focused on. Since the current SVG Tiny engines from major manufacturers are not free and we can not get the specific implementation details, so the study and optimization in this paper is based on the self-developed SVG Tiny engine. A partial rendering algorithm and circular arc contour algorithm are proposed through the analysis of rendering module in the engine. The contrastive testing shows that these two algorithms can improve the efficiency of the engine and achieve satisfactory results.

2. Related work

2D vector graphics drawing is a complex and time-consuming process, and there are mainly two ways to achieve it on mobile devices: software implementations such as the polygon filling algorithm [5], and interface implementations such as the OpenVG [6] proposed by Khronos organization. However, very few devices support OpenVG, and some of them use the way of software simulation to implement OpenVG with low efficiency. Another way is to use 3D graphics rendering interface, such as OpenGL ES[7] to achieve the 2D vector graphics rendering. But because 2D and 3D interfaces are not unified, it is difficult to achieve it. Rendering is the most time-consuming step[5] to draw 2D vector graphics, and the current optimization for it focuses on the following aspects: optimization of rendering processes [8], partial rendering, efficient vector graphics filling algorithm [5] and hardware accelerating [9][10][11].

Partial rendering is an important optimization method. The rendering speed can be greatly improved by calculating the area that needs to change and only rendering this area. The previous ideas of partial rendering are mostly used for rendering windows and controls in UI systems, and very few studies aim at the partial rendering of the 2D vector graphics. The partial rendering strategy [5] has a certain effect,
but there also exists the issue of partial rendering inefficiencies when the distance of the invalid areas between different elements is far. To solve this problem, we redesigns the partial rendering strategy, and the proposed algorithm has greatly improved the performance compared with the original algorithm.

An contour algorithm for stroke font is proposed [12] and can be applied well to the contour algorithm for other graphics after modification, but it does not mention the arc contour algorithm. The arc contouring is an important part of drawing 2D vector graphics and its quality directly influences the speed of the polygon filling. Currently, the arc contouring is mainly achieved in two ways: one is directly based on the arc characteristic equation. The drawback is that in addition to the contouring of bezier curve, it also needs to support contouring of other quadratic curves. This method affects the performance and is not conducive to migration. The other is to convert the arc into a bezier curve before performing the contouring operation and its drawback is that there is a certain error after conversion. But its advantage is that an very high accuracy can be reached when the polar angle is not large, and it can simplify low-level interface during implementation and is very easily combined with other open interfaces such as OpenVG. So it is used by current most mobile devices. There are a lot of existing algorithms[13][14] for fitting quadratic curve like circle and ellipse with bezier curve, and many scholars have done in-depth studies[15][16]. The popular approach at present is to convert a circle or ellipse into four cubic bezier curves, with each quadrant represented by a cubic bezier curve. It is found that within the arc angle in the range of 0 to $\frac{\pi}{2}$, the error can reach the range from $1.96 \times 10^{-4}$ to $2.73 \times 10^{-4}$ by using bezier curve to fit arc[17], and such an error is essentially imperceptible for mobile devices with small screen. For the method of fitting arbitrary circular arc with bezier curve, some description has been made [15]. But in order to meet the accuracy requirements, higher degree bezier curve is needed and the amount of calculation is a great burden for mobile devices. This paper presents a simple and effective algorithm which uses cubic bezier curve to fit arbitrary circular arc. The method achieves satisfactory results in the testing on mobile devices.

3. Partial rendering

The algorithm[5] divides the rendering into two steps Calculator and Render. The Calculator step calculates the invalid area of each element, and ultimately composites all the invalid areas into the invalid area of document. The Render step renders the contents in the invalid area of document formed in the Calculator step. But the algorithm has a big flaw. If the invalid areas of elements are scattered, then the final invalid area of document will be large (shown in Figure 1), which is against the original intention of partial rendering. From another point of view, it is because that only one partial rendering area is saved in the scene and it contains the invalid areas of all elements.

![Figure 1. The partial rendering area in [5]](image)

So we propose a partial rendering algorithm called PRA which comes from the improvement of the algorithm [5]. PRA dynamically traces the invalid areas of elements in the Calculator step to produce multiple reasonable partial rendering areas, and then renders them in turn in Render step. This method effectively avoids the defect of too large partial rendering area in the original algorithm and improves the rendering efficiency. The improved algorithm is described as follows:

Step 1 Assuming that the screen will eventually be divided into not more than $n$ partial rendering areas, the area of each partial rendering area is initialized to zero. Then starts to traverse the elements that need to be drawn in the scene tree.
Step 2 Obtains the invalid area of an element, and composites this invalid area with each partial rendering area. Takes the new area whose difference with the original partial rendering area is the minimum after compositing to replace the original partial rendering area until the completion of traversing the scene tree.

Step 3 At this point, there are \(i\) \((i \leq n)\) partial rendering areas whose areas are not zero.

Steps 3.1 If any two partial rendering areas are intersected, then composites them. The aim is that if the intersected partial rendering areas are not composited, then the intersection parts of them will be rendered twice. Supposing there are many semitransparent elements in the scene, secondary rendering will get the wrong result.

Step 3.2 Repeats Step 3.1 until all remaining partial rendering areas cannot continue to be composited. At this moment, there are \(k\) \((k \leq i)\) partial rendering areas in the scene.

Step 4 Traverses the scene tree to render the elements in the partial rendering areas. When drawing a specific element in the scene, checks in turn whether its bounding box is intersected with the \(k\) partial rendering areas. If they are intersected, then draws its parts in the partial rendering area. Step 3.1 ensures that there are no intersected partial rendering areas, so repeated rendering would not happen.

Next, we will interpret above algorithm using several figures. Figure 2 shows the initial state of the algorithm, and the areas of all the partial rendering areas are zero, which are not represented in the scene. Then, the engine begins to traverse the scene tree. Figure 3 represents the state of calculating the invalid area of the first element, and this area is used as the partial rendering area 1.

Figure 2. The initial state

Figure 3. Calculate the first invalid area

Figure 4 represents the state of calculating the invalid area of the second element. At the moment, if the invalid area of element B is composited with that of element A, a large invalid area will be produced, which is also the shortcoming of the algorithm [5]. According to Step 2, the invalid area of element B will be composited with another partial rendering area which is in the initial state, and this area will be used as the partial rendering area 2.

Figure 4. Calculate the second invalid area

Then the engine starts to calculate the invalid area of the third element, and compositing calculation is carried out between the invalid area of element C and each partial rendering area in turn (including the partial rendering areas in the initial state). Eventually, it is found that if the invalid area of element C is composited with the partial rendering area 1, the area difference between the new area and the original partial rendering area 1 is minimum. It shows that the composition of the invalid area of
element C into the partial rendering area 1 will increase a smaller drawing area, so it is the most optimal way, as shown in Figure 5.

![Figure 5. Calculate the invalid area of element C](image)

![Figure 5. Composite the invalid area into partial rendering area 1](image)

After traversing the scene tree, there are three partial rendering areas in the scene. The partial rendering area 1 is intersected with the partial rendering area 3 (shown in Figure 6-a), according to Step 3, composition is carried out between these two areas (shown in Figure 6-b). Finally, the engine only needs to draw the elements in the partial rendering area 1 and 2.

![Figure 6. Calculate the third invalid area and composite it into the partial rendering area 1](image)

When drawing the elements in these partial rendering areas, however, it requires special treatment if there are clipping areas in the scene. For example, when the scene contains animation or textArea elements, these elements need to reflect the clipping effect in rendering, the size of clipping box of these elements has been specified in the definition. We need to calculate the intersection areas of the clipping box with the partial rendering areas, and only the parts of these elements in the intersection areas need to be drawn (shown in Figure 7). After this kind of special drawing is completed, the original partial areas need to be restored for the drawing of other elements.

![Figure 7. Handle elements with clipping area](image)

The parameter $n$ in Step 1 determines the number of partial rendering areas, and there are several ways to set the value of $n$.

Method One: If there are $m$ nodes as the animation target in the scene, then $n \leq \log_2 m$. In order to facilitate the storage of the scene tree, the scene tree is generally designed as the form of binary tree. Supposing that there are altogether $d$ nodes in the scene tree, then the depth of the tree is $\log_2 d$. When designing the scene, experienced developers generally believed the node at the same depth is logically associated with each other. In principle, a partial rendering area is assumed for the nodes at the same depth. However, in general, there does not necessarily exist the animation target nodes at every depth.
Consequently, the number of nodes as the animation target is calculated, resulting in a more satisfactory value of \( n \).

Method Two: The value of \( n \) depends on the number of elements with the depth of 1, provided that the elements are drawable or contain drawable sub-elements.

Method three: After a new invalid area of element has been calculated, we composite the invalid area with the existing partial rendering areas. If the area variation after composition is larger than this invalid area as an independent partial rendering area, it is necessary to create a new partial rendering area.

The first two methods use the static allocation to create \( n \) partial rendering areas, and the drawback is that the irrational scene file structure will result in a waste of memory. While the third method utilizes the dynamic allocation approach, whose disadvantage is that an excess of piecemeal elements in the scene will generate a lot of smaller partial rendering areas, thus increasing the engine cost when rendering. Therefore, these three methods are integrated in the practical implementation. We firstly utilize the first or second method to calculate the maximum value of \( n \) and then adopt the third method to allocate dynamically the partial rendering area. The number of created partial rendering areas is no more than \( n \). When \( n \) is equal to 1, ACA is equivalent to that algorithm [5].

4. Fitting circular arc with bezier curve

When using bezier curve to fit circular arc, if no changing the parameters of bezier basis or control matrix, we only can get an approximate arc. In order to draw a precise circular arc, it is necessary to add \( \sin \) and \( \cos \) parameters to the bezier basis and control matrix, which may increase the computational complexity on mobile devices, thereby reducing the performance. In this paper, we present an algorithm for fitting circular arc called arc contour algorithm(ACA). ACA calculates the control points of cubic bezier curve according to the midpoint rule. ACA has low complexity and is suitable for mobile devices. ACA is suitable for the case when the arc angle \( \alpha \) is in the range of 0 to \( \pi \) and can get best accuracy when \( \alpha \) is in the range of 0 to \( \pi/2 \). If \( \alpha \) is greater than \( \pi \), the arc will be subdivided into two cubic bezier curves.

In order to convert a circular arc whose angle is in the range of 0 to \( \pi \) to a cubic bezier curve, it is necessary to know the coordinates of the endpoints \( p_0 \) and \( p_3 \), and then to calculate the coordinates of the other two control points \( p_1 \) and \( p_2 \) according to the geometric knowledge. The algorithm is shown as follows:

Step1. The cubic bezier curve is described with the following parameterized polynomial equation:

\[
p(t) = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3 \quad 0 \leq t \leq 1
\]

(1)

According to the vector synthesis theorem and tangential theorem, the point \( p_1 \) and \( p_2 \) can be expressed as following(shown in Figure 8):

\[
p_1 = p_0 + k \times (y_0-x_0)
\]

(2)

\[
p_2 = p_3 + k \times (-y_3,x_3).
\]

(3)

Step2. According to the equation (1), when \( t = 1/2 \), we get the point with the coordinate of \((p_0+3p_1+3p_2+p_3)/8\), denoted as \( A \). According to the casteljau algorithm, it is the intersection point of arc bisector and the arc. The length between point \( A \) and the center point \( O \) is the radius, denoted as \( r \). The coordinate of the intersection point of arc bisector and line \( p_0p_3 \) is \((p_0+p_3)/2\), denoted as \( B \). And the length between point \( B \) and point \( O \) is denoted as \( |B| \), as shown in Figure 9. Suppose the coordinate of \( p_0 \) is \((x_0,y_0)\) and the coordinate of \( p_3 \) is \((x_3,y_3)\), we get:

\[
|B| = \frac{|p_0 + p_3|}{2} = \frac{\sqrt{(x_0 + x_3)^2 + (y_0 + y_3)^2}}{2}
\]

(4)

Figure 9 shows that \( A/ r = B / |B| \), we have:
\[
\frac{(p_o + 3p_1 + 3p_2 + p_3)}{8 \times r} = \frac{(p_o + p_1)}{2 \times |B|}
\]

(5)

Step 3. Put the coordinates of \(p_1\) and \(p_2\) into the equation (5), we get:

\[
\begin{align*}
4 \times (r - |B|) \times ((x_0 + x_3) \times (y_0 + y_3)) &= k \times ((y_0 - y_3), (x_3 - x_0)) \\
3 \times |B|
\end{align*}
\]

(6)

According to equation (6), two solutions of \(k\) can be obtained:

\[
\begin{align*}
k &= \frac{4 \times (r - |B|) \times (x_0 + x_3)}{3 \times |B| \times (y_0 - y_3)} \quad |y_3 - y_0| > |x_3 - x_0| \\
k &= \frac{4 \times (r - |B|) \times (y_0 + y_3)}{3 \times |B| \times (x_3 - x_0)} \quad |y_3 - y_0| \leq |x_3 - x_0|
\end{align*}
\]

According to the coordinates of \(p_0\) and \(p_3\), we can calculate the value of \(k\) and then solve the coordinates of \(p_1\) and \(p_2\) by put \(k\) into equation (2) and (3). Finally, we can carry out the arc contouring calculation according to equation (1).

**Figure 8.** The control points of cubic bezier curve

**Figure 9.** The position of the key points

When contouring, if the length of circular arc is very short, it is not necessary to generate too much stroke endpoints. If the arc length is extremely long, it is also necessary to control the number of stroke endpoints: if the distance between stroke endpoints is too short, additional computational burden may emerge and there is no obvious effect on smoothing the arc. If the distance is too long, obvious edges and corners will be generated in the arc. Therefore, we specifies that the number of stroke endpoints of arbitrary circular arc is no more than 40, and the distance between two stroke endpoints is no less than 3 pixels. Supposing \(n\) is the number of output stroke endpoints after contouring, the following method can be used to control the value of \(n\):

\[
m = \max(x_0, x_1, x_2, x_3) - \min(x_0, x_1, x_2, x_3) + \max(y_0, y_1, y_2, y_3) - \min(y_0, y_1, y_2, y_3)
\]

(5)

\[
\begin{align*}
\{ & n = m \quad m \leq 40 \\
& n = 40 \quad m > 40
\end{align*}
\]

In above equation, \(x_0, x_1, x_2, x_3\) and \(y_0, y_1, y_2, y_3\) respectively represent the coordinates of x and y of control points \(p_0, p_1, p_2, p_3\). After calculating the value of \(n\), we can know the interval of parameter \(t\) in equation (1) and calculate the coordinates of all stroke endpoints.

5. Tests

In order to evaluate our partial rendering algorithm, we use the SVG Tiny 1.2 test suites beta3[18] published on September 12, 2008 by W3C as test cases. Test environment is HTC G7 (resolution of
480 * 800 and CPU of 1GHZ), HTC G8 (resolution of 240 * 320 and CPU of 528HZ) and HTC G9 handset (resolution of 320 * 480 and CPU of 600HZ), which are representatives of the screen resolution of current mainstream mobile phones. To make a comparison with PRA, we also install opera mobile 11 which can run the test cases on these phones. We use the rendering fluency FPS (frames per second) to evaluate the efficiency of our partial rendering algorithm. The test results of opera mobile 11 are obtained mainly by visual inspection and timing approach because frame rate information is not provided by opera mobile 11. Due to the large number of test cases, it is incapable to list all the data, so here only two typical cases are cited.

Table 1. The test results of animate-elem-30-t.svg

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC G8</td>
<td>18 FPS</td>
<td>25 FPS</td>
<td>24 FPS</td>
<td>&lt; 15 FPS</td>
</tr>
<tr>
<td>HTC G9</td>
<td>13 FPS</td>
<td>15 FPS</td>
<td>20 FPS</td>
<td>&lt; 10 FPS</td>
</tr>
<tr>
<td>HTC G7</td>
<td>12 FPS</td>
<td>14 FPS</td>
<td>20 FPS</td>
<td>&lt; 10 FPS</td>
</tr>
</tbody>
</table>

Table 2. The test results of animate-elem-37-t.svg

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC G8</td>
<td>12 FPS</td>
<td>14 FPS</td>
<td>15 FPS</td>
<td>&lt; 10 FPS</td>
</tr>
<tr>
<td>HTC G9</td>
<td>10 FPS</td>
<td>12 FPS</td>
<td>15 FPS</td>
<td>&lt; 10 FPS</td>
</tr>
<tr>
<td>HTC G7</td>
<td>9 FPS</td>
<td>10 FPS</td>
<td>14 FPS</td>
<td>&lt; 10 FPS</td>
</tr>
</tbody>
</table>

Table 1 and Table 2 show that in the case of small resolution, all the algorithms can achieve satisfactory rendering fluency. But with the increase in screen resolution, even though the CPU performance has been largely improved, the rendering fluency of full-screen rendering, rendering algorithm in [5] and opera mobile11 all have been reduced a lot, which is because there is no partial rendering technology or algorithm is not excellent. On the contrary, all the test cases using PRA have obtained satisfactory fluency.

For the test of arbitrary arc fitting, we make a comparison with the traditional drawing method using arc equation, as well as the algorithm in [15]. The test focuses on the contouring time of arc with the unit of millisecond, and the test platform is the HTC G8 phone. α refers to the arc angle, r refers to the length of arc radius, rc refers to the number of repeated calculation, and the test time is the cumulative time of repeated contouring calculation for 5 times, as shown in Table 3.

Table 3. The test results of arbitrary arc fitting

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>α=π/2; r=10px;rc=5</td>
<td>10ms</td>
<td>8ms</td>
<td>8ms</td>
</tr>
<tr>
<td>α=π; r=10px;rc=5</td>
<td>19ms</td>
<td>15ms</td>
<td>14ms</td>
</tr>
<tr>
<td>α=2π; r=10px;rc=5</td>
<td>42ms</td>
<td>40ms</td>
<td>33ms</td>
</tr>
<tr>
<td>α=π/2; r=50px;rc=5</td>
<td>52ms</td>
<td>40ms</td>
<td>38ms</td>
</tr>
<tr>
<td>α=π; r=50px;rc=5</td>
<td>108ms</td>
<td>82ms</td>
<td>74ms</td>
</tr>
<tr>
<td>α=2π; r=50px;rc=5</td>
<td>220ms</td>
<td>212ms</td>
<td>152ms</td>
</tr>
</tbody>
</table>

For the traditional method, each scan line will generate a vertex, and the contouring time shows a linear increase with the arc length growth. The algorithm [15] can utilize the cubic bezier curve when the arc angle is smaller, and its contouring time is equal with that of ACA. However, with the increase in the arc angle, the algorithm[15] represents the arc by the higher degree bezier curve, so the contouring time is significantly prolonged. For ACA, the arc with larger angle will be divided into two sections, and each one will be represented by a cubic bezier curve. Compared with other algorithms, the contouring time of arc drawing shows significant optimization.

6. Conclusions

The core of SVG Tiny and its associated technologies is still the parsing, layout and rendering of the scene. By analyzing the scene rendering, this paper presents a dynamic partial rendering algorithm, which can also be applied in other vector graphics drawing technologies through some improvement, such as the canvas technology of html5, Java 2D technology, etc. We also propose the Bezier fitting optimization technology of arbitrary arc which can improve the low drawing efficiency of the traditional technology on mobile devices, and is apt to combine with other open vector graphics...
rendering interfaces such as OpenVG. Many aspects in the optimization of SVG Tiny engine still need to be explored, such as using the mobile cloud computing capabilities to simplify parsing or execute script, using the OpenGLES hardware accelerating to achieve the 2D vector graphics drawing, using the matrix characteristics of coordinate transformation to optimize the graphics drawing, and the bezier fitting of arbitrary quadratic curve, etc. The follow-up studies will focus on these points.

7. References