An Optimal Browser Design for GPU Utilisation

Ekioh Ltd, Cambridge UK.
flow@ekioh.com

Abstract

This paper examines the way in which browsers exploit the benefits of hardware acceleration and asserts that most traditional browser designs currently fail to unlock the full benefits of the GPU. The GPU usage of Ekioh's multi-threaded HTML browser, Flow, is outlined and the performance differences are highlighted. It is concluded that, in order to approach the performance necessary to render an animated 4K UI on a cost effective embedded system, more focused techniques such as those developed within the Flow browser need to be exploited.

Introduction

The forerunner of the GPU, the 2D hardware blitter, was introduced in the 1980s and was designed to process graphics pixmaps without CPU intervention. Blitters are still in widespread use today in embedded hardware for 2D graphics acceleration and composition, though are more recently supplemented with a GPU. The first recognisable GPU was launched in the late 1990s although the term was not in widespread use until 1999[1]. Since then, GPUs have evolved to become a vital component in all graphically rich products from mobile phones to gaming PCs.

Initially, GPU development was driven by the desktop computing and gaming console markets where dedicated fast memory is used and the whole screen is re-rendered on every frame. Mobile and tablet evolution has accelerated the growth of the embedded GPU market; on these, the screen is broken down into a series of tiles which can be processed individually[2].

Within the embedded market, the GPU and CPU are usually packaged together with other key application-specific functionality in a 'System on Chip' (SoC) device. Most SoCs are connected to a single block of memory which is shared by the CPU and GPU making memory performance more of a limiting factor; if either processor dominates the memory accesses, the other's performance will be impaired. This is exacerbated by consumer electronics price pressures which commonly lead to the use of lower speed memory.

The rendering approach of traditional browsers

On a dynamic page, redraw needs to be repeated whenever any part of the page changes. Browsers try to minimise the amount of work that the CPU has to do in order to maintain performance. The elements that require updating are marked as 'dirty'. The browser then calculates the 'dirty areas' which are the minimal bounding areas of these dirty elements. It then triggers a repaint of these dirty areas that includes all the dirty elements and the parts of any other elements that overlap them. Most browsers use the CPU to paint the representation of these elements into a pixmap.

Typically, the next frame to be displayed is created by taking the previous frame and overwriting the changes, as shown in Figure 1. A hardware accelerator (2D blitter or GPU) is used to combine the previous frame and partial pixmap data into the next frame freeing the CPU for other tasks.

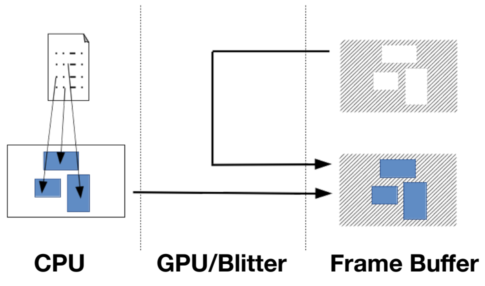

Figure 1

Accelerated compositing

An alternative method is accelerated compositing which offers significant benefits as the size of the dirty areas become large.

Accelerated compositing is the technique of grouping parts of the render tree into layers which do not interact and so can be manipulated independently. These layers are rasterised into separate pixmaps.

When anything changes, the hardware accelerator composites the pixmaps representing each layer into the frame buffer, as shown in Figure 2. The composition of layers always covers the entire screen and so there is no need to access a copy of the previous frame. Changes to some HTML/CSS properties, such as transform and opacity, can be handled by the hardware accelerator during this composting phase avoiding the need to repaint the contents of the layers.

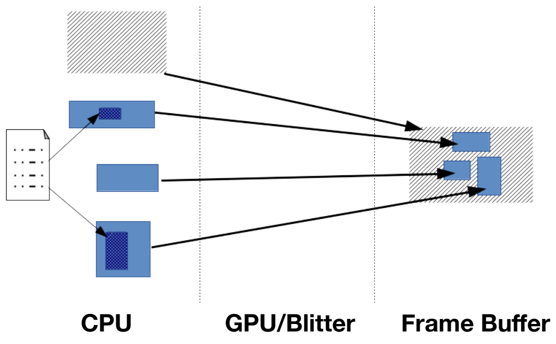

Figure 2

Traditional browsers use heuristics to determine when accelerated compositing will improve performance. These heuristics are not infallible; seemingly insignificant content changes can result in the browser returning to non-composited rendering or causing repaints of the contents of the layers. This can lead to a dramatic loss of rendering performance and introduce the need for potentially lengthy content optimisation cycles to force rendering back onto the hardware accelerated path whilst, at the same time, keeping memory usage to a minimum. In an effort to force the use of accelerated compositing, content developers often employ tricks, for example, 3D transforms using *translateZ(0)*[3].

Accelerated compositing is essentially a tradeoff between memory and CPU usage. Since the GPU handles the compositing of the pixmaps into the frame buffer the benefits for pages, where animations are achieved solely using transforms and opacity, can be significant. Whilst tricks, like those described above, are available to content developers to increase rendering performance, there is a corresponding increase in memory consumption which may have undesirable effects elsewhere in the system.

Using the GPU

Although GPUs offer far more flexibility than blitters, browsers have been relatively slow to embrace their full benefits. The interrelated nature of browser design makes it complex to redesign a core component, such as rendering, to exploit vastly different hardware capabilities. As a result, most browser designs still treat the GPU as an enhanced blitter with some additional transform capabilities and use it mainly for accelerated compositing.

With accelerated compositing, pixmap layers are still painted by the CPU. These pixmaps are then copied to the GPU. Mechanisms often exist on embedded GPUs to pass pointers to blocks of shared memory to avoid the need for copying, however the optimal memory format for the GPU can be different to that of the CPU. When this is the case, the use of 'zero copy textures' can reduce rendering performance as the GPU is forced to perform data format conversions on every frame update. As display resolutions increase, the memory requirements of the cached pixmaps will also increase which can have a material impact on platform costs.

Browser providers have been looking to move beyond the simple 'GPU as a blitter' scenario for some time. Using the GPU, as opposed to the CPU, to handle painting is known as GPU rasterisation. Some browsers include the option to enable GPU rasterisation using extended graphics libraries that pass the painting tasks to the GPU. These libraries cater for a wide range of usage scenarios and hence offer very flexible, generic, APIs. Unfortunately, the flexibility of these APIs is overcomplicated for the use cases required by HTML and this leads to sub-optimal use of the GPU leading to reduced rendering performance.

GPU performance comes from being able to efficiently execute a large volume of similar operations batched together, feeding pipelines to ensure they are optimally filled. The flexibility provided by these extended graphics libraries often leads to a failure to keep the GPU's pipelines full which significantly reduces its effectiveness.

Maximising the benefit of the GPU

The Flow browser takes a different approach. Instead of focusing upon a set of flexible graphics primitives to be implemented on the GPU, Flow's design focuses solely upon the requirements of an HTML browser and matches those to GPU capabilities.

The characteristics of HTML lend themselves well to GPU acceleration. Elements are essentially rectangular, naturally grid aligned, and rarely use anti-aliasing. Since GPUs process triangles, two can be used for each rectangular HTML element.

A highly efficient HTML-specific solution can be implemented where display commands are translated into sets of triangles that are submitted to the GPU. The GPU is then responsible for converting these triangles into pixels on a hardware graphics surface, and displaying the results on-screen.

The set of graphics primitives required for HTML are also relatively small which means that an HTML-specific GPU accelerated graphics API is a realistic goal.

The GPU rasterisation process happens completely asynchronously to the CPU. This means that the CPU can start processing subsequent tasks such as scripting and layout before the painting task is complete. GPU rasterisation also changes the optimisation-reward balance. On the CPU, performance optimisations are achieved by only updating those areas of the screen which have changed, whereas on the GPU it is optimal to follow the lead of the gaming industry and update the entire screen every frame. Whilst it may seem counter intuitive to re-paint every pixel on each frame, in most cases, the processing cost of redrawing the whole screen is lower than cost of the dirty region calculations which would be needed on the CPU to manage partial updates.

Using GPU rasterisation also means that Flow removes the constraints that lead to content developers forcing the use of hardware accelerated rendering. Avoiding these cached layers also means considerable memory savings and, as there are no significant memory exchanges between the CPU & GPU, performance is improved. Furthermore animation performance can be maintained even when those animations involve layout changes that require significant CPU involvement. Flow's multithreaded layout engine helps maximise the amount of layout that can be achieved within the 16.6ms available between frames to ensure 60fps animation rates.

The specific nature of Flow's graphics API makes it possible to create large batches of graphics commands and data, which leads to more optimal pipelining and increased GPU efficiency. Lightening the load on the CPU/GPU interactions also minimises the potential of GPU stalling, where the GPU is idle and waiting for the next instruction from the CPU. This, in turn, leads to better performance scaling as each increment in GPU performance will deliver a matching increase in browser rendering performance.

Using GPU rasterisation for the painting task delivers a small benefit in overall performance when browsing static web pages but, for dynamic UI applications, the performance improvements can be dramatic.

The results

To verify the results, a series of side-by-side performance tests were conducted comparing Flow, Safari, Chrome and Firefox on a quad core MacBook Pro using an Nvidia Geforce GT 650M GPU.

Where rendering optimisation options were available on the other browsers, these were also tested to ensure a fair comparison; both Chrome and Firefox have experimental options to enable a level of GPU rasterisation so tests were carried out both with these features enabled and with the default configuration (disabled). In an extreme rendering test involving whole screen animations Flow was shown to be over twice the speed of its nearest rival, see Figure 3.

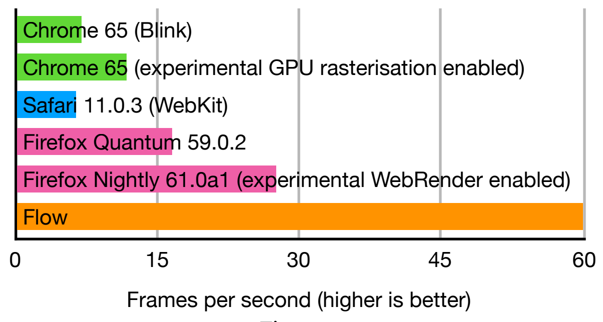

Figure 3

Using 'The Man In Blue' HTML animation benchmark[4] with the number of rendered objects doubled to 1000 to drive up the intensity of the test, Flow's GPU rasterisation maintains a clear performance advantage as shown in Figure 4.

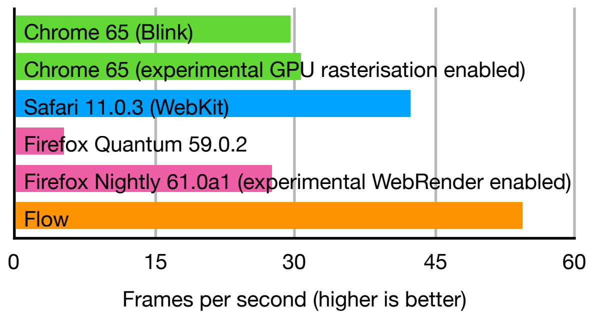

Figure 4

Testing on a GPU-enabled quad core ARM based set top box (4 x 3K DMIPS per core) showed a similar pattern of results when compared against the box's default WebKit based browser:

- Flow achieved 44fps in the extreme rendering stress test compared to 5fps on WebKit.
- 'The Man In Blue' animation benchmark (with 500 particles), was somewhat closer, with Flow achieving 17fps compared to 12fps on WebKit.

Looking towards a 4K UI

Current products promoting 4K capabilities refer to their video resolutions rather than their UI resolutions which are still HD. A 4K UI (3840 pixels x 2160 pixels) has four times the number of pixels as an HD UI (1920 x 1080). 4K's increased resolution presents a number of new and exciting UI design opportunities, but brings with it a significant increase in the amount of data to be manipulated.

The processing impact of a 4K UI on a traditional browser could be reduced if animations were limited to only use techniques which can be achieved using accelerated compositing and therefore require little or no CPU interaction. The likely four fold increase in cached pixmap memory requirements would, however, have a material impact on product cost.

As discussed earlier, accelerated compositing heuristics can be failure prone; with a 4K UI, software fallback performance would be unacceptably poor so content optimisation requirements could become onerous. Restricting the way in which the 4K UI is implemented, such that accelerated compositing could be guaranteed, would also limit the exciting UI design opportunities that this increased resolution provides. Given this, it is unlikely that rich 4K UIs would be feasible on next generation hardware using traditional browser technology.

Flow's GPU rasterisation does not suffer from the increased memory requirements or design restrictions imposed by accelerated compositing. Flow's rendering performance is governed by the throughput of the GPU and therefore benefits directly from each new generation of embedded device which often includes a big step forwards in GPU performance.

As manufacturers increase the number of GPU cores in their devices, a greater number of tiles can be rendered in parallel. This, along with other silicon improvements, provide the increased GPU throughput necessary for a 4K UI. Because Flow's rasterisation is not limited by the CPU, it is able to realise the full potential of the GPU to deliver graphically rich 4K UIs.

Summary

The results show that Flow's use of GPU rasterisation can at least double animation performance and reduces graphics memory requirements. This, coupled with its use of multithreaded layout, make 4K UIs a practical possibility on high end embedded silicon today and it is projected that, using Flow, the next generation of silicon will deliver 4K UIs on mass market consumer electronics products.

About Ekioh

Ekioh designs and develops rendering software for a wide range of consumer products and resource constrained platforms. The company's products provide the high performance, compact footprint and flexibility necessary to address the wide range of application and UI rendering challenges presented by the consumer electronics and embedded systems industries. Best known for its TV and set top box solutions, Ekioh browsers are deployed in tens of millions of products in over 30 countries around the world.

Ekioh's engineering team brings together expertise in graphical systems, embedded software, set top box, TV silicon and robust software design. Using this combination of skills, fuelled by intense customer focus, Ekioh's product quality and customer support are second to none.

References

1. "NVIDIA Launches the World's First Graphics Processing Unit: GeForce 256". Nvidia. 31 August 1999
http://www.nvidia.com/object/IO_20020111_5424.html
2. A look at the PowerVR graphics architecture: Tile-based rendering
https://www.imgtec.com/blog/a-look-at-the-powervr-graphics-architecture-tile-based-rendering/
3. "CSS GPU Animation: Doing It Right". Smashing Magazine
https://www.smashingmagazine.com/2016/12/gpu-animation-doing-it-right/
4. The Man In Blue Animation Benchmark
http://themaninblue.com/experiment/AnimationBenchmark/html/