Abstract

Nowadays, visual effect is become more and more common in computer user interface. Since web page is one of the most extensive way to communicate between human and computer, a good design with proper visual effect is much more user-friendly. We can see there are need of the creating visual effect or animation in web environment.

In the past, the developing of the web animation is very hard, because the history of the web technology is very old and there are many kind of web browsers which may behave very differently with the same segment of code. It is a very tedious job when testing the code compatibility between web browsers. The resolve this problem, most web programmer choose to use a framework to develop their project rather than create every from the ground. We can see there are many JavaScript framework or library already. Such as jQuery, which is the most common JavaScript framework for working with the DOM (Document Object Model) Element in the web page.

This animation framework is aimed to help web developers to simplify the development process and also adapt the new technology and provide a corresponding fallback for the browsers do not supporting or may behave in different way. The framework can save the work of the web developer or web designer for testing the code compatibility between web browsers and creating simple web animation excellent performance with just few line of codes.

Table of Contents

Introduction

Background information

Web is the most ordinary platform interacting with people to provide or exchange information nowadays. Having the advantage of different web technologies, from single page web application to Ajax web application, the web platform can provide a more user friendly interface to user. The more user friendly interface the more graphical element involved in it such as animation. One JavaScript framework – jQuery can do this, but it is not design for animation and the result is in poor performance. The other solution is CSS animation. It provides a very convenient way to create high performance or even hardware accelerated animation for developer, but the drawback is the animation cannot be too complicated (for example rotate and scale the element at the same time with different time ease), because CSS-animation binds all the animation into a single transformation (Myth Busting: CSS Animations vs. JavaScript, 2014). With the JavaScript animation, the animation can be more complication and creative, since JavaScript can dynamically generate animation. A JavaScript animation framework can make the creation of web animation become more easy and creative.

This project is aimed to create a client based JavaScript animation framework for web developer or designer to create efficient canvas animation easily. To achieve the highest performance, the framework will make use of the "web worker", a JavaScript API which is allowed JavaScript program running in parallel, to process the animation such as key frame, collision handling and infection between rendering object. The advantage of using web worker is it can offload the heavy loading job (long loop, recursion call) to other threads and maintain the UI thread be responsive to user. This will keep the animation smoothly and preserve the frame rate.

Why focus on mobile device?

As the processing power of mobile device is becoming more and more powerful today, and different operating system arise in the market such as iOS, Android and Windows Phone 8. This makes developer need to spend more time and effort to develop a dedicated version for each operating system. Webapp is the solution to cater this problem. Webapp is a single development, multi-platform application, which can save the development cost. Developers do not need to deal with the low level API of different operating system. Although developers still need to deal with the web browser compatibility problem, an animation can save the work for developer. Mobile device all has limited processing power compare with Personal computer. This makes an efficient animation framework become important. This is the idea of QoS come from.

Project aims and objectives

This project is aimed to create a JavaScript Animation Framework which is compatible to most of the modern browser in mobile device and using the new technology standard with the fallback support for the old browser.

The difficulty of this project is how the synchronize the rendering process and other worker thread and at the same time to minimize the reaction time of the animation. Imagine that, if we send a huge batch of data to the worker thread, it will take a long time to process the data, but if we send many small batch of data to the worker thread, it will keep interrupt the main thread when it finished the process. This definitely will affect the rendering performance. The key point is how to take the balance of this.

The framework use the web technology Canvas. Which is very different from the traditional web page. Canvas is mainly used for image processing and rendering, but if web developers or designers want to use canvas to create an animation or game, it is a very tedious job for them. Because canvas does not come with the native API which support animation. Web developers need to creation the animation frame by frame and also need to aware the performance impact in different approach.

Literature review

Feedback Control Read-Time Scheduling: Framework, Modelling, and Algorithms (Lu, Stankovic, Son, & Tao, 2002)

The control theory and the feedback control system in this paper had give me a glance of the system can monitor itself based on the past performance and determine how to handle or process the tasks in the coming queue. This is how the QoS system in this framework is designed.

Perfect spatial hashing (Lefebvre & Hoppe, 2006)

In traditional, hashing is a process to map data into a list of buckets, and the mapping is based on the hashing algorithm and most likely is random.

Spatial hashing used a special hashing algorithm to map the objects according to the their coordination and dimension to a 2D or 3D array of buckets. Object occupied same location space will be mapped to the same bucket. Using this technique can quickly to determine the list of near objects or collision detection, without using brute force to calculation all of the object combination.

In this framework, spatial hashing is used to determine the dirty objects when a canvas cleared a regions. This can speed up the rendering pipeline without redraw every objects on the canvas on each frame.

High Performance JavaScript (Zakas, 2010)

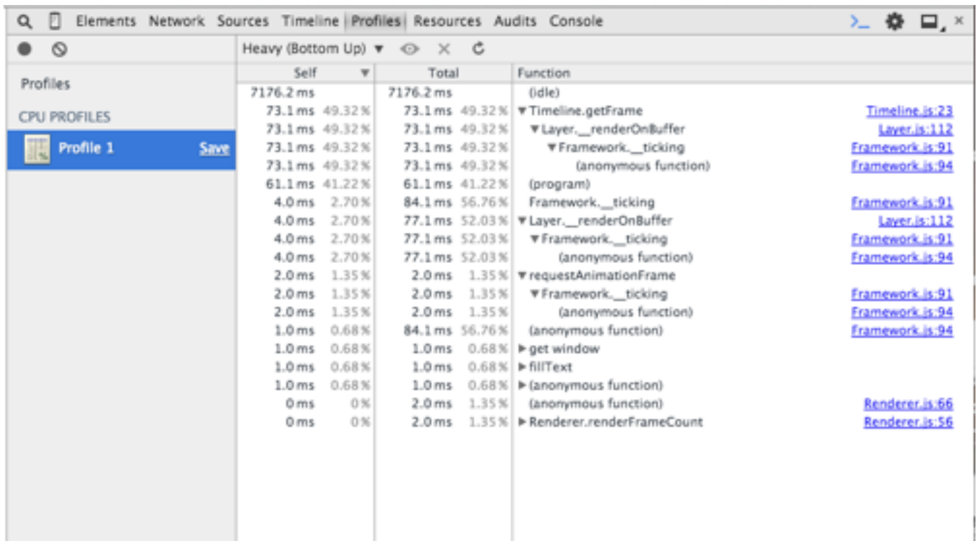
This book listed a lot of techniques to improve the performance of JavaScript and avoid the performance traps.

This book is very useful for JavaScript programmer. It helped me to find the best way to loop through an array and let me know the performance different of Object vs Array.

Profiling JavaScript Performance (Performance Tips for JavaScript in V8, 2012)

Throughout the development of this project, The Chrome profiler is very important to use to identify the performance of the script or code. This give programmer a hint to optimize their program.

This is one of the CPU profile of this project. Using the CPU profiler we can see that the most CPU consuming function is the "getFrame" in the "Timeline.js: 32". We can use these information to optimize the program and improve the performance.



```
        Self                Total         Function
 7176.2 ms            7176.2 ms           (idle)
   73.1 ms  49.32 %     73.1 ms  49.32 %  ▼Timeline.getFrame                 Timeline.js:23
   73.1 ms  49.32 %     73.1 ms  49.32 %    ▼Layer.__renderOnBuffer          Layer.js:112
   73.1 ms  49.32 %     73.1 ms  49.32 %      ▼Framework.__ticking           Framework.js:91
   73.1 ms  49.32 %     73.1 ms  49.32 %          (anonymous function)       Framework.js:94
   61.1 ms  41.22 %     61.1 ms  41.22 %   (program)
    4.0 ms   2.70 %     84.1 ms  56.76 %   Framework.__ticking               Framework.js:91
    4.0 ms   2.70 %     77.1 ms  52.03 %  ▼Layer.__renderOnBuffer           Layer.js:112
    4.0 ms   2.70 %     77.1 ms  52.03 %    ▼Framework.__ticking             Framework.js:91
    4.0 ms   2.70 %     77.1 ms  52.03 %        (anonymous function)         Framework.js:94
    2.0 ms   1.35 %      2.0 ms   1.35 %  ▼requestAnimationFrame
    2.0 ms   1.35 %      2.0 ms   1.35 %    ▼Framework.__ticking             Framework.js:91
    2.0 ms   1.35 %      2.0 ms   1.35 %        (anonymous function)         Framework.js:94
    1.0 ms   0.68 %     84.1 ms  56.76 %   (anonymous function)              Framework.js:94
    1.0 ms   0.68 %      1.0 ms   0.68 %  ▶get window
    1.0 ms   0.68 %      1.0 ms   0.68 %  ▶fillText
    1.0 ms   0.68 %      1.0 ms   0.68 %  ▶(anonymous function)
      0 ms      0 %      2.0 ms   1.35 %   (anonymous function)              Renderer.js:66
      0 ms      0 %      2.0 ms   1.35 %  ▶Renderer.renderFrameCount         Renderer.js:56
```

Figure 1

Improving HTML5 Canvas Performance (Improving HTML5 Canvas Performance, 2011)

In search of the way to make the canvas rendering become faster, this web page (HTML5) recommended to create a buffer canvas to pre-render the object before render it on the main canvas.

This web page also suggested using multiple layered canvases for complex scenes. This inspired is project using the layer architecture to minimize the unnecessary object redraw.

The Basics of Web Workers (The Basics of Web Workers, 2010)

Web Worker is the JavaScript API which is allow JavaScript running in parallel. This tutorial demonstrate how to create a web work and the limitation of it.

This tutorial also demonstrate the way to communicate between web worker and the main JavaScript program. There is a faster way more than cloning the whole object to the worker (JavaScript Work cannot share data or object). Transferable object can be transferred into the web worker and this is much faster than cloning the object.

The secret to silky smooth JavaScript animation!

The web page (The secret to silky smooth JavaScript animation!, 2012) suggested a correct way to synchronize the animation to the browser rendering step. This is very different from the traditional way using "setTimeout" or "setInterval" as the counter to trigger the animation rendering.

The traditional "setTimeout" or "setInterval" only set the constant frame rate and cannot be change after that. In the Google 2015 I/O Event, google suggested that in increase the Android rendering frame rate increase to 200Hz (Amadeo, 2015). Which means in future the 60Hz of the rendering frame rate would be too slow for those devices.

Using the "requestAnimationFrame" API is an adaptive approach to synchronize the rendering frame rate to the browser and this can improve the rendering performance because it eliminate the redrawing overhead.

Performance Tips for JavaScript in V8

This web page (Performance Tips for JavaScript in V8, 2012) suggested the programming guideline to increase the performance in JavaScript V8 execution engine.

For example: avoid using hidden classes, because this will cause the object type changing during the runtime and this will effect the code execution performance.

Static Memory JavaScript with Object Pools (Static Memory JavaScript with Object Pools, 2013)

The garage collection mechanism of JavaScript is automatically done by the system and the process of this will pause the whole program and cause a poor experience to users. Frontend developer have no right to control the process of garbage collection and when to release the memory space which are no longer needed. The solution of this is using object pool to reserve the unused space, so that reuse the allocated memory space if needed. This can control the process of garbage collection. Because Pre-allocated object pool can keep the amount of active memory used by the thread to be a constant. When some object is no longer need, it will return the memory to the object pool instead of return the system memory directly. This can reduce the change of the system conducting garbage collection actively.

Proposed design, solution, system

The main design of the system is using web worker to offload the calculation work to other thread and keep the main thread focus on the rendering and respond to user action and also to maintain the QoS level when the computer cannot handle all of the rendering or processing task. This is the class diagram. This framework is using requireJS as the module loader to include different modules.

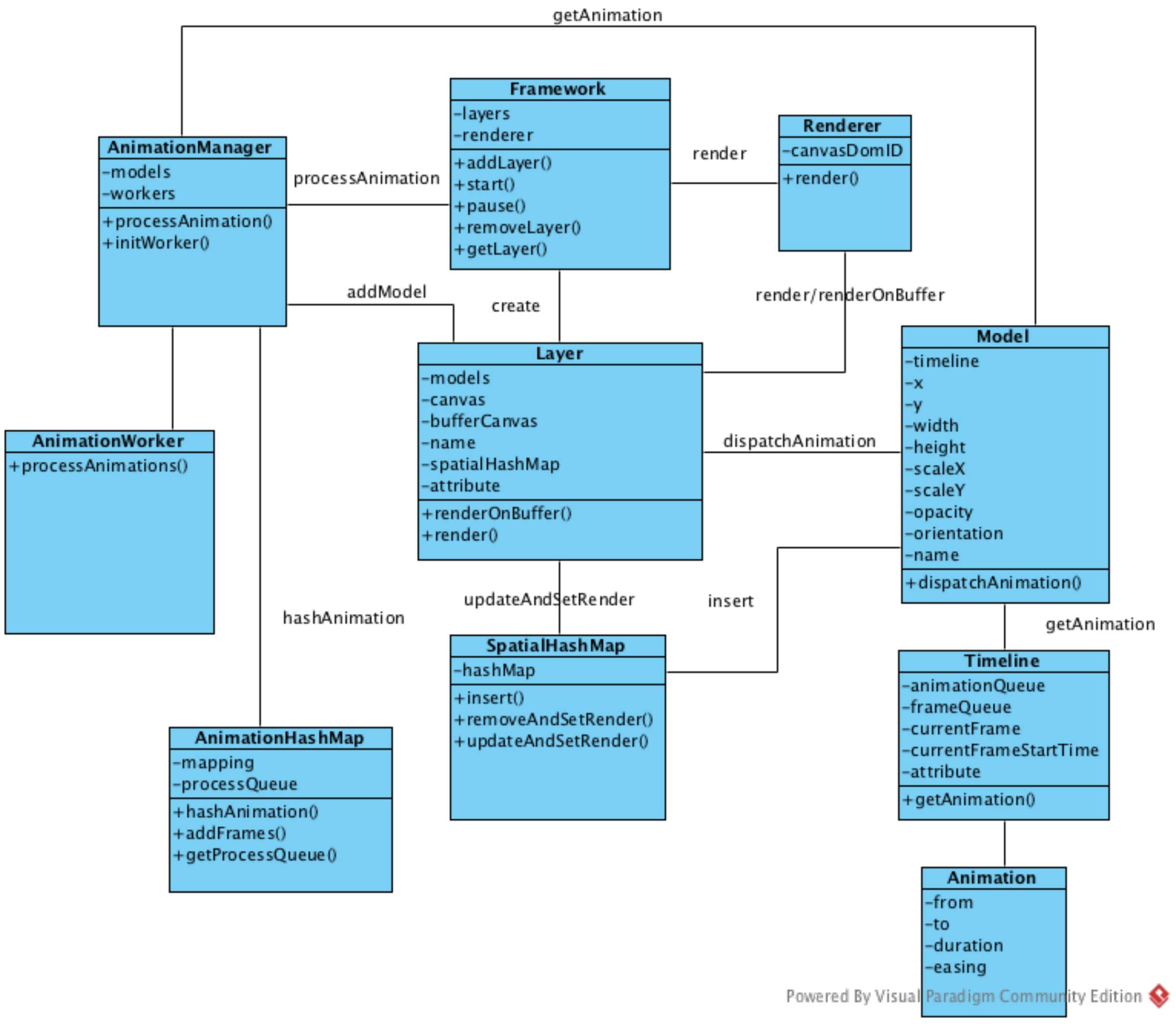


Figure 2

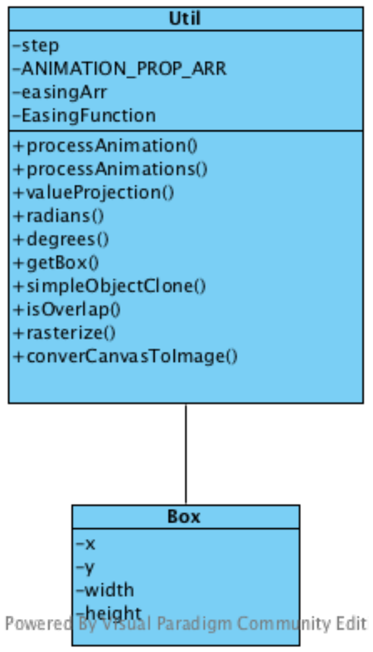

Util
-step
-ANIMATION_PROP_ARR
-easingArr
-EasingFunction
+processAnimation()
+processAnimations()
+valueProjection()
+radians()
+degrees()
+getBox()
+simpleObjectClone()
+isOverlap()
+rasterize()
+converCanvasToImage()
Box
-x
-y
-width
-height

Figure 3

Rendering System

This framework provided two render method and each of them is suitable for different scenario. The rendering methods are the traditional brute force rendering and spatial rendering. Since the canvas system in HTML5 and JavaScript is only store the canvas image in a pixel array and there is no way to remove a image completely from the canvas without clearing the canvas. Because the images may overlap with each others, their shape can be irregular and also contain transparent part. If clear one of the image dirtily, the canvas may result in containing incomplete images or models and affect the visual effect seriously.

The traditional way to deal with this problem is clearing the whole canvas and redraw all of the elements on the canvas each frame. This is very waste of processing power and harm for the battery time. If the number of objects on the canvas is not many, this approach still feasible and can produce smooth animation. But if the number of objects is increased drastically, the processing power may not be able to handle the redrawing all of the elements on each frame and the animation starting became unstable and dropping frame.

The ideal rendering method the only redraw the objects need to be update and keep other objects untouched. But this is hardly or even impossible to achieve in JavaScript Canvas. Because the JavaScript Canvas is only an array of pixels and only stored one colour on each pixel. There are no layer, image cache or image buffer mechanism in the JavaScript Canvas. If a pixel is overlapped by other objects, it is unable to recover or restore to the original state.

Alternative way to update the canvas – Spatial Hashing

This framework is able to use Spatial hashing to determine the near objects which may need to be redrawn. It is good for models well arranged or small portion of the models is need to redraw. In these cases, spatial hashing can decrease the number of models need to redraw significantly. If the models are arranged in random position or more than 50% of the models need to be redrawn at the same time, spatial hashing cannot improve the performances and even worse the frame redraw time. Because when the number of models need to be redrawn is increasing, there are more models hashing overhead is added. When the redraw models portion increase to 50% or more and the models is arranged randomly, it is nearly the whole screen will be redrawn and the processing time for the hashing is wasted. Eventually the number of redraw draw objects is near the method clearing and redraw all of the canvas elements.

Details implementation

The original JavaScript implementation is retrieved from here. (Christer, 2014) I have done some modification in order to fit in the framework and improve the performance by changing the mapping from associate array (Object) to 2D array.

The basic idea is taking the coordinate of the top left corner and the bottom right corner and using the bitwise shift operator to map the coordinates to an array index.

Take is as an example. An Object A {x: 100, y: 50, width: 50, height 50}. The top left corner is (100, 50) and the bottom right corner is (150, 100). We shift all coordinates to right by 5. The hashed coordinates is (3, 1) and (4, 3). After getting the hashed coordinates we can insert A into the following buckets: [3][1], [3][2], [3],[3], [4][1], [4][2], [4][3] and this is the insertion procedure.

Usage

If the Object A above is removed from the canvas or being updated and we need to update the screen without redraw the whole canvas, we can make use of the spatial hashing to find out the affected objects when clearing the rectangle region of Object A. We only need to apply the hashing algorithm again and find out the corresponding buckets of Object A and set the rendering flag of all other objects in the buckets to on. After that the renderer will add these objects in to the rendering queue and follow the z-index to rendering these objects and complete the dirty region cause by clearing Object A.

Quality of Service

The marketing of the mobile phones or tablet is very wide. The hardware can be very different. The number of cores in CPU can be ranged from 1 to 8 cores and the memory is started from 1GB to 4GB. This kind of hardware diversity make the mobile application developer of web programmer may need to give up some of the feature of their program or web page, so that it can be run smoothly in all kind of the mobile devices and dominate the market. They cannot give up all of the awesome feature of effect because of the low end market. But it is a tedious job for the programmer to optimize their programme and select suitable feature for all of the hardware and operating system.

In mobile devices, battery life also is one of the main concern. The processor on the mobile devices may limit the processing power itself to prolong the battery life or prevent the CPU is being overheat. Therefore the processing power of the CPU may vary under different conditions. A Quality of Service (QoS) mechanism is designed for this scenario.

QoS is applied widely in the area of computer networking. In computer networking, the QoS is guarantee the high priority network package can be deliver to their destination on time. The QoS mechanism in this framework is similar. When the model is created in the framework, they can be assigned a corresponding QoS level, which is ranged from 0 to 9. The smaller QoS level is more important and the zero level is always render to the canvas. Models with other QoS level will be rendered in the consideration of the average frame rate or the rendering deadline.

Quality of Service System Design

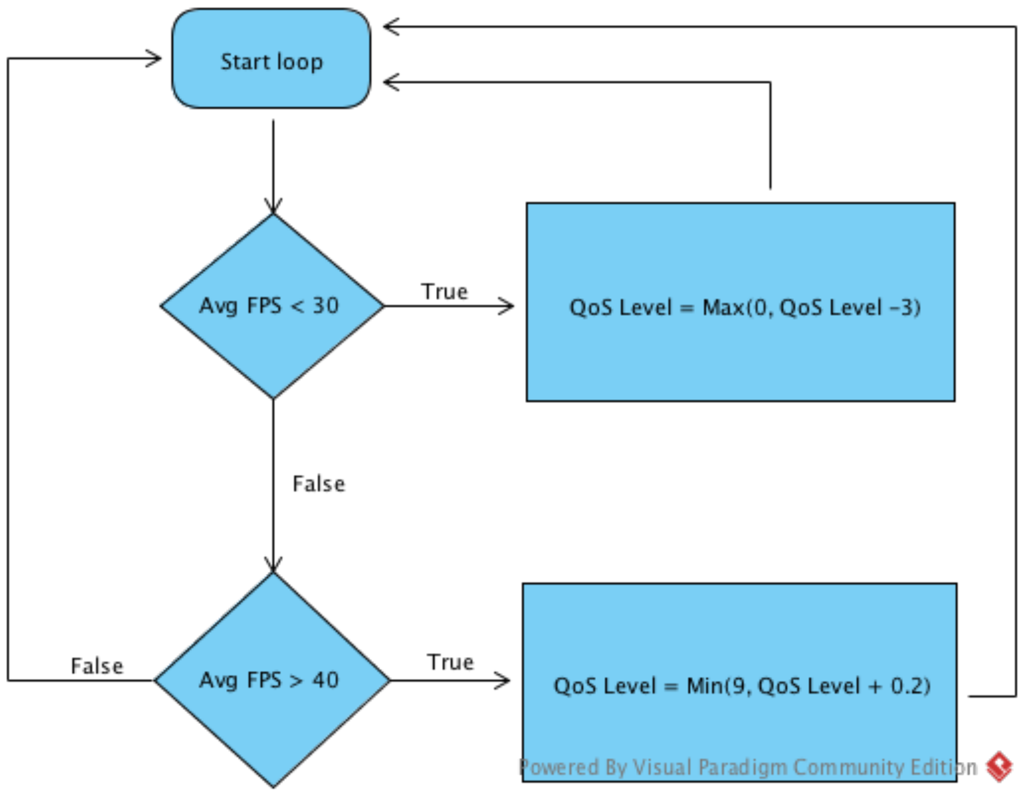

Start loop
Avg FPS < 30
True
QoS Level = Max(0, QoS Level -3)
False
Avg FPS > 40
True
False
QoS Level = Min(9, QoS Level + 0.2)
Powered By Visual Paradigm Community Edition

Figure 4

The above flow chart shows the details implementation of the QoS system. Where "Avg FPS" is the average FPS of the last second. During the rendering of each frame, it will make use of the rendering data of the last frame to determine the QoS level of the coming frame. The purpose of the asymmetric design to change the QoS Level is wants to stabilize the rendering frame rate and prevent the frame rate being oscillated.

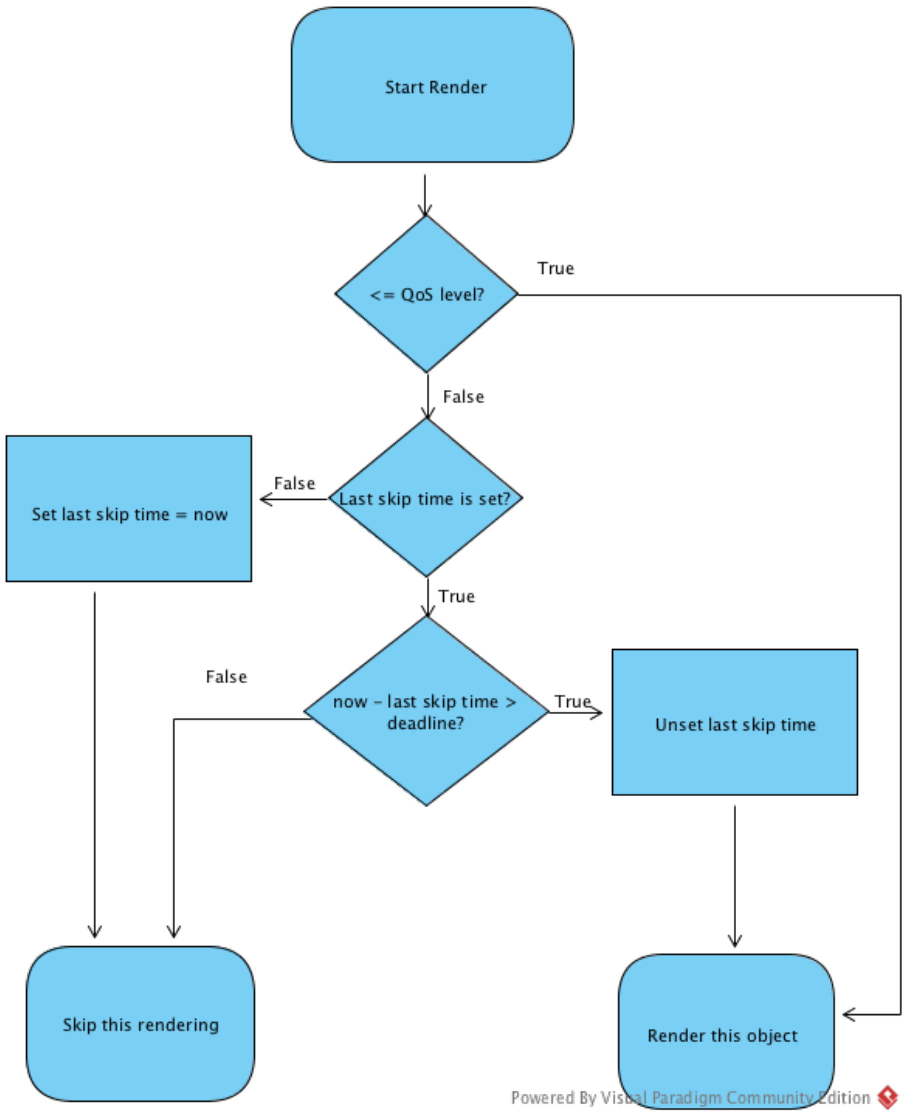


Figure 5

This flow chart shows the process of determine render an object. First it will check the object QoS Level if it is smaller or equal the current QoS Level, it will pass to render directly. If it is lager then the current QoS level, it will be passed to check the last skipped time and the deadline. If the last skipped time is not set, it will set the last skipped time to now and skip this rendering. If the last skipped time is set, it will compare the duration between now and the last skipped time. If the time lapse is lesser than the deadline, it will skip this object otherwise it will reset the last skip time and send add this object to to rendering queue.

Detailed methodology and Implementation

Framework - The main component

The major component of this framework. The duty of it is control the rendering flow and sequence. In order to synchronize the animation frames to the browser rendering. The framework does not use the traditional browser delay execute API “setTimeout” or “setInterval”. Because these APIs call cannot synchronize the rendering or page reflow event of the browser, and may cause the unnecessary screen re-render.

This component uses the “requestAnimationFrame” (CreativeJS, 2012) browser API to control the rendering flow. This API will be invoked when the browser is ready to render and this can make the rendering process synchronise with the browser rendering event and minimize the redraw overhead.

The framework also wrapped the “requestAnimationFrame” and invoked the fallback function, please refer to the appendix for the detail implementation.

The responsibility of the component is initialize the Renderer, AnimationManager and provide an access for user to create a new layer.

AnimationHashMap

This class is store the mapping of the animation and the frames value.

How it work

Each of the animation contains the start value, end value, duration and the easing function. For example: “move an object from x=100 to x=150 in 1 second in a linear movement.” We can translate this into a “delta instruction” such as “50 0 1000” Where the 50 is the delta value, 0 is the index of the easing function and 1000 is the duration. Let the frame rate is 60fps, and this instruction will create 60 values in this deltaArr [0, 0.833, 1.667, …., 50]. This array is general for all of the animation with the delta is 50 in linear easing and the duration is within 1 second.

During the rendering process, the new value is “old value + deltaArr[(timelapse/duration) * fps]”. There are still some calculation has to be done during the rendering time to calculate the new value but it could save more time during the rendering stage if the easing function is very complicated.

AnimationManager

This is the main component to communicate with the web worker. The job of this class is fetching the information of animation from models and passing it to the worker to process. In case the web worker fails to initialize or the client browser is not support the web worker, this component will invoke the fallback function and process the animation information in the main thread.

During the each rendering cycle, the Framework class will trigger the AnimationManager to fetch new animation from all of the models and hash the animations by the AnimationHashMap. The AnimationHashMap will translate the animations into a "delta instruction (hashed key)".

```
{
        from:{
                x: 100,
                y: 120
        },
        to:{
                x: 150,
                y: 100
        },
        duration: 1000,
        easing: "linear"
}
```

```
{
        x: "50 0 1000",
        y: "-20 0 1000"
}
```

Figure 6

The above example show that the AnimationHashMap will translate the animation into a "delta instruction". These instructions means "x" will increase by 50 with the easing index 0 in 1 second and "y" will decrease by 20 with the easing index 0 in 1 second. After getting the "delta instruction" the AnimationHashMap will try to looking these instructions into the mapping. If the instruction can not found in the mapping, it will return a frame with duration -1 and AnimationManager will sent the instructions to web workers to process. When the web worker completed the frames, the AnimationManager will update the timeline of the model and also the AnimationHashMap for later instruction reference. If the instruction is founded in the mapping, the AnimationHashMap with return the frames to the AnimationManager and append the frames to the timeline of the model.

Layer

Each layer can be referred to a physical Canvas DOM Element overlapping with each other and contain the model mapping and the timeline. Different layers can be stored different type of models. This can minimize the re-rendering of the scene.

Take using the framework to create a game as an example. The background layer is the layer changed the less. Normal layer may contain the character or other object which changed constantly with the same background. The UI layer may contain the timer or score board. Which is changed frequently. Separate those object into three or more layers can minimize the object to be re-render and increase the performance. (HTML5 Rocks).

The layer also hold the physical canvas reference and execute the actual rendering job. During rendering cycle, the layer will check the through the objects list the evoke the frameDispatch() call of the model, and the if the model need to be updated in the canvas, it will set the "isActive" to true to indicate the coordinate of the model is updated and ready to be render.

When the model "isActive" property is true, the system will clear the old and new region occupied by the model, push these regions into the dirty regions list and using the spatial mapping to set overlapped objects to the rendering queue and also update the mapping.

After all the models is checked, the system will render the models in the rendering queue one by one according to their z-index and preserve the models' order. When all the models is rendered into the buffer canvas, the system will clear the dirty regions list on the actual canvas and copy the pixels inside the dirty regions list form the buffer to the actual canvas. Because only the images inside the dirty regions is need to be redraw and preserved the ordering.

Renderer

Renderer control and synchronize the two rendering process. The first is render on buffer, which is a off screen canvas used for buffering the pixel on each layer. The renderer will render each object on the buffer first. When all of the objects is rendered in the buffer, the renderer will render the image of the buffer to the original canvas. (HTML5, 2011)

The advantages of use a buffer to rendering are it can improve the rendering performance and synchronize the rendering process. Since buffer canvas is a off screen canvas, which means rendering on the buffer will not affect the browser rendering and does not cause any UI reflow. The performance of the off screen canvas is much better than the normal canvas. This can save the overhead between the browser renderer and the JavaScript renderer.

The other object of the renderer is the QoS level control. Since the renderer control the rendering flow and sequence, it can detect the rendering time of each frame during the rendering cycle and adjust the QoS level using the QoS feedback control.

Worker

Web worker is a class which is running in different thread, to be specify, it is running in a different JavaScript VM. Therefore, it does has some limitation when using it. For example it does not has the “document” object and cannot access any object that could cause the reflow or re-rendering. (HTML5, 2010)

Due to these limitation, the web work is designed to perform calculation task, such as processing the key frame, pre-calculate the object coordinate.

The implementation of the worker is only forward the call to other class and send back the data when the calculation is done. The function used by the worker is store in the “Util” class. It can be shared by all of the other class of the framework. This is because the worker class only is a wrapper to wrap the call and process it the other thread in the purpose of parallel computing. In case of the browser of JavaScript runtime is not support web worker, the processing function still need to be available to other classes to take over and complete the job.

The JSON object in the JavaScript is very handy and useful for structured data. It can easily to form meaningful data and pass the reference or property. But transferring JSON object between web worker or main thread this very slow, and not feasible using JSON to transfer large frames data. To get rigid of this, typed array (Flanagan, 2011) this used in this framework for passing data between web worker. Typed array is similar with the array type in C. It has specific type and length and cannot be changed after the array is created. Typed array is very low level implementation in the JavaScript runtime and therefore can be transferred between web worker and also the accessing speed is very fast. It is very suitable for the framework which need to process and transfer large amount of data.

Since the typed array is a plain array and dose not contain any structure. The web worker has to implement the structure in the array. For example when transferring the “delta instruction” into the web worker, the index of the element is represent it type and cannot be changed. When the web worker completed the job and send back the frame to the main thread, it also has to send back the “delta instruction” so that the frame can be reconstructed at the main thread from the typed array.

Timeline

This is used to store a sequence of animation queue or processed frames. When an animation is added to the model, the animation will be passed into the timeline and stored in the animation queue for later processed by the AnimationManager.

During the rendering time, the timeline will records the time delta between the first key frame is rendered. Using the time delta, the timeline determine which frame in the frame sequence should be render as the coming frame.

The timeline will keep trace the current frames set and rotate to the next frames set when the current animation is completed.

This framework support to way to add animation, “append mode” and “immediate mode”. Append mode will append the animation into the end of the animation queue and set the animation “from” property to the “to” property of the last animation in the queue. This is making the animation to be continuous. When the new animation is add in “immediate mode”, the timeline will clear all the existing animation and frames queue and shift the animation to the current frames when it is processed by the web worker and ready to render.

Animation

The animation object contained the information describe the object movement and store in the following format (Figure 6). This example represented the animation to translate an object from (x:100, y:200) to (x:150, y:100) in 1 second and the easing is linear. More property can be add in the animation, such as orientation, opacity, scaleX and scaleY. This can make the animation to be more enriched. Refer to the appendix to see more supported easing function.

```
{
        from:{
                x: 100,
                y: 120
        },
        to:{
                x: 150,
                y: 100
        },
        duration: 1000,
        easing: “linear”
}
```

Figure 7

Model

Structure

The width and height property is the coordinate of the model and the “img” property is created by the model constructor. It will base on the URL to reload the image and store to the “img” property.

The other property is used of the animation, such as rotation, scale and change of opacity. Each of the object will be associate with an array indexed by the object name and zIndex.

The model also containing the property at different time. They are “current”, “last” and “final”. The “current” and “last” property is used to clearing the old and new regions when the model need to be updated. The final property is used to append an animation or setting the model manually by the model method “set()”.

There are also some other property for the QoS system, such as the QoS Level and the last skipped time.

Constructor

The model constructor allow developer to create any model need to be animated in the canvas. The current framework only implemented the image model constructor. The other type of model will be implemented in the following stage.

Different type of the model will be parsed by different constructor. For the image type, the constructor will load the image and with the width and height property in the model. If the type of the model is an arbitrary canvas model, it will be draw on a static off screen canvas and store it to the “img” property. The reason is this can save the time of rasterize the image when drawing it to the canvas on each frame.

Util

This is a static class for storing shared method or function. For example the easing function, getBox function and simpleObjectClone function.

Put every together

User use the “Framework” as a start point to create a “layer”, and add a “model” to the “layer”. Each “model” has their own “timeline”, inside the “timeline” there are two queue, “animationQueue” and “frameQueue”. When user add an “animation” to a “model”, the “animation” will be stored into the “animationQueue”. When an “animation” has been processed, it become a serial of “frames” and it will be stored into the “frameQueue”. In short, the frames are compiled/processed animation which are serials of values.

In each ticking or rendering cycle, there are mainly two functions will be invoked, and they are “renderer.render()”, and “AnimationManager.processAnimation()”. The “renderer.render()” has 3 mission, execute the QoS control code, “layer.renderOnBuffer()“ and “layerrenderOnCanvas()”. The QoS control code is the flow chart mentioned above.

The function of “renderOnBuffer()” of layer is loop through all the models and fire the “dispatchAniamtion()” to trigger the “model” request new frame from timeline and update the model’s property if needed. If the model is updated, it will set the “isActive” flag to on and the “layer” will clear the the old and new region and add these regions to the “dirtyRegions” of the model and using the function of “sptialHashingMappig.updateAndSetNearModelRerender(model)”. This function will set the related models’ “needRender” flag to on and update the mapping.

When all models has be invoked the “dispatchAnimation()” and updated the mapping, the layer will loop through the models list again and render the models to the “bufferCanvas” with the “needRender” flag is on.

After the “renderOnBuffer” is completed, the “layer.renderOnCanvas()” will be invoked, and this function will loop through the “dirtyRegions” list to copy the pixels inside each region to the canvas. This is all the rendering pipeline.

The remain is “AnimationManager.processAnimation(). This function will loop through all the models (on all layers) and compile the “Animation” into “Frames”. The step is get the first animation from the “model.timeline.animationQueue”, pass the animation to the “AnimationHashMap” the “AnimationHashMap” will translate the “Animation” into “delta instruction” and looking into the mapping” If all the “Animation” is find in the mapping, it will return the complete frame to the “AnimationManager” and it will push the “frame” into the “frameQueue”. If there is some animation is not found in the mapping, it will return a frame with the duration -1 and add the “delta instruction” to the “processQueue”. When all the models are looped though, the “AnimationManager” will send to “processQueue” to the web workers to process. The “processQueue” is a list of missing mapping of the “delta instruction”. The job of the web worker is complete the mapping. When the web workers is completed and send back the result to the “AnimationManager”, it will push the result to the mapping in “AnimationHashMap” for later reference. In next rendering cycle, the “AnimationManager” will send the same animation missed in last cycle to the “AnimationHashMap” and this time the animation will successfully compiled into frames and appended into the “framesQueue” in the “timelime” and ready for updated the model and rendering.

This is the Animation processing pipeline.

Statistical result

Designed Test Cases

Table 1

System Setting	QoS	Spatial Hashing	Deadline
1. QoS SH 40	Enable	Enable	40ms
2. QoS SH 20	Enable	Enable	20ms
3. QoS SH	Enable	Enable	Disable
4. SH	Disable	Enable	Disable
5. Baseline	Disable	Disable	Disable

The QoS is trying to skip some of the object rendering in order to reduce the number of rendering job, so it can only produce stable an animation when Spatial Hashing rendering is enabled. The images with the setting QoS enable and Spatial Hashing disable will be flashing or disappear. It is not a feasible setting and solution for making animation, therefore not tested in here.

These test case is generated by JavaScript and run one by one automatically. The first setting is ran two time to "warm up" the CPU. Because the new CPU have a turbo boost feature, it can speed up the CPU for a few second until the heat generated is over the heats ink can bear. The first setting is not count into the result. Each of the setting is last 10 second.

Case 1

Render 3500 Objects, each QoS level has 350 objects, 500 Objects is moving, each of the object is 50x50px, Screen size 1920x1080

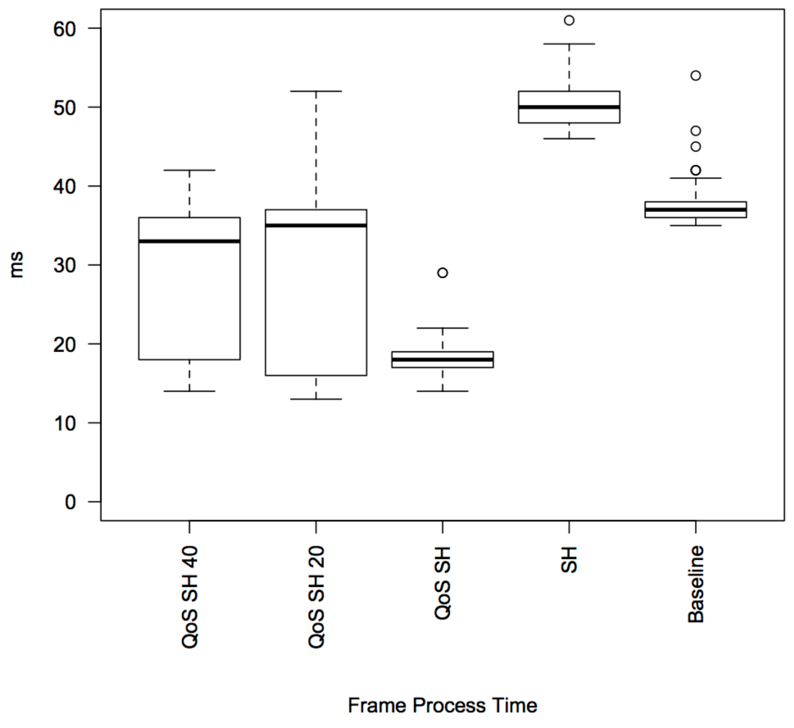


Figure 8

QoS SH: In this graph, we can see that rendering using Spatial hashing enable, QoS enable and deadline disable had the best performance. But since the deadline was disabled, some objects which was out of the QoS level was not animated. This setting in fact did not have 500 objects moving at each frame, but still produced an acceptable animation. The key objects (such as objects with QoS Level = 0) still animated and preserve the key functionality.

SH: This setting only with Spatial hashing enabled has the worse performance. Because it has to redraw all of the 500 moving objects and the objects overlapped. The objects need to be redrawn are distributed all over the screen, therefore the drawing queue is similar with the baseline. The hashing overhead decreased the performance and even worse than the baseline. This setting is not usable and will produce choppy animation. Should prevent using this setting for the scene have many objects moving at the same time.

QoS SH 40/QoS SH 20: These two settings have the deadline enabled and the average frame rendering time is around 33ms. Which is the QoS trying to maintain the processing time not longer then this. If the processing time of each frame is more than 33ms, the animation will become choppy and affect the visual effect. The data distribution of these settings is dispersed compare with others. This is because the QoS system has the response time to set the suitable QoS level, and also need to handle the objects rendering deadline. These make the processing job of each frame become unpredictable. The setting with the deadline 40ms has a slightly better performance compare with the setting with deadline 20ms. Which is because the longer deadline can the renderer skip more rendering task in each frame.

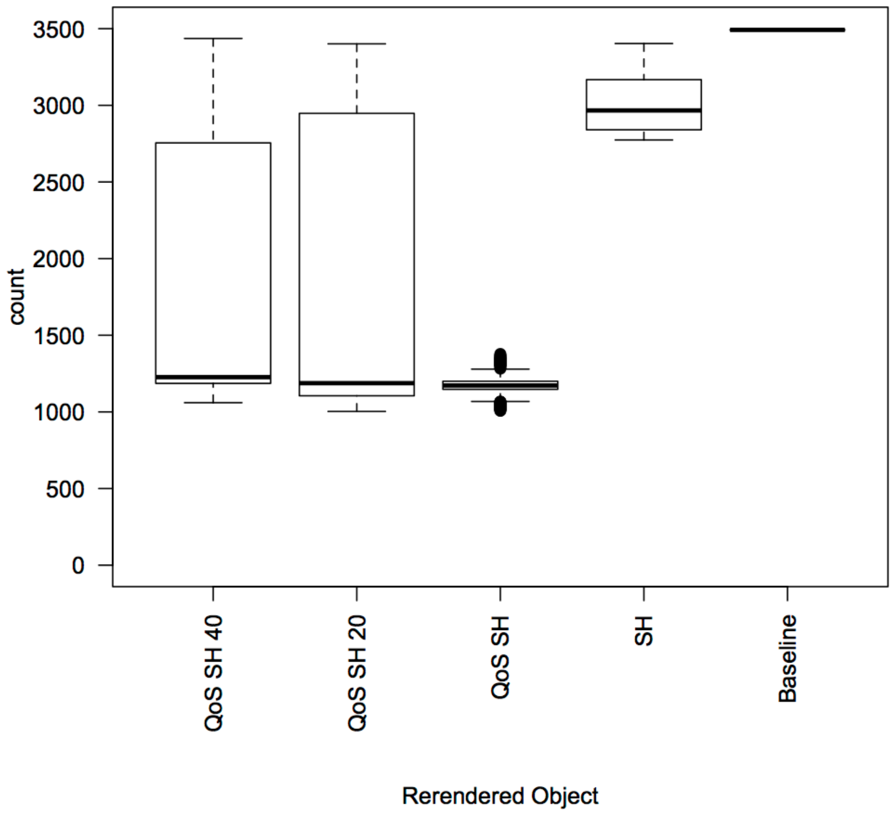


Figure 9

In this graph, we can see that the QoS can decrease the number of objects redraw on the each frame significantly. Without the QoS enable, moving 500 objects will cause around 3000 objects redraw, this was only 14% better then the brute force method. But the QoS can minimize the number of redraw count to around 1300 objects. Which was 62% better that the brute force rendering method.

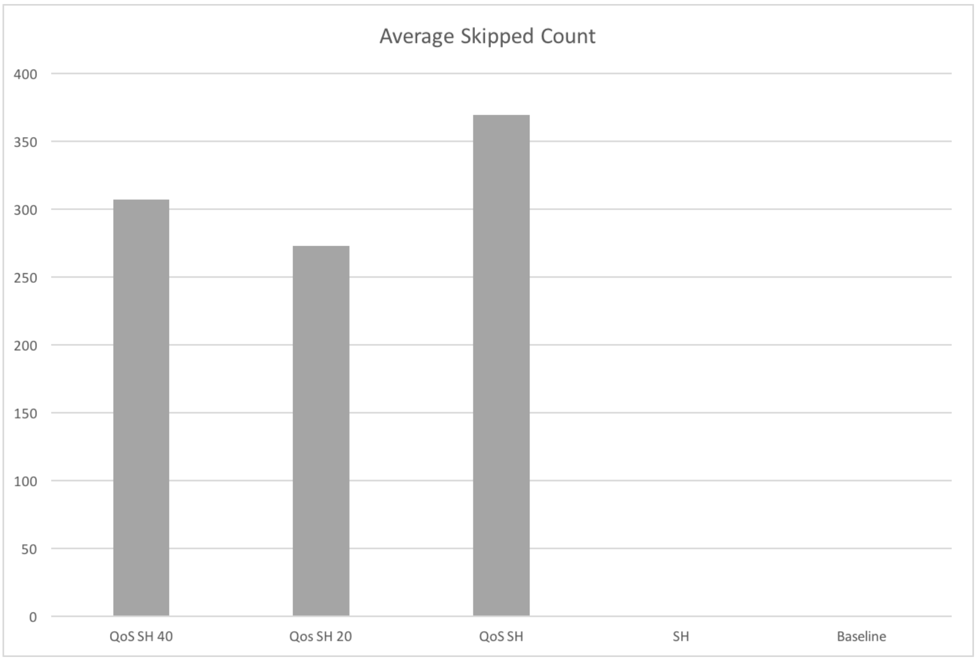


Figure 10

This graph showed the average objects rendering was skipped of each frame. We can see that the longer deadline the more objects rendering were skipped. When the QoS was disabled, every objects had been rendered and no object was skipped.

To summarize, this test case showed that the spatial hashing has a poor performance when the there are many objects need to be updated on the canvas at the same time. Because it create a lot of overhead during the hashing process and this affect the performance seriously. If implemented the QoS system at the same time with the spatial hashing rendering method, the QoS system can eliminate or skipping some of the objects update and decrease the overhead created by the hashing and this can make spatial hashing still can produce acceptable animation under the extreme condition.

Case 2

Render 3500 Objects, each QoS level has 350 objects, 50 Objects is moving, each of the object is 50x50px, Screen size 1920x1080

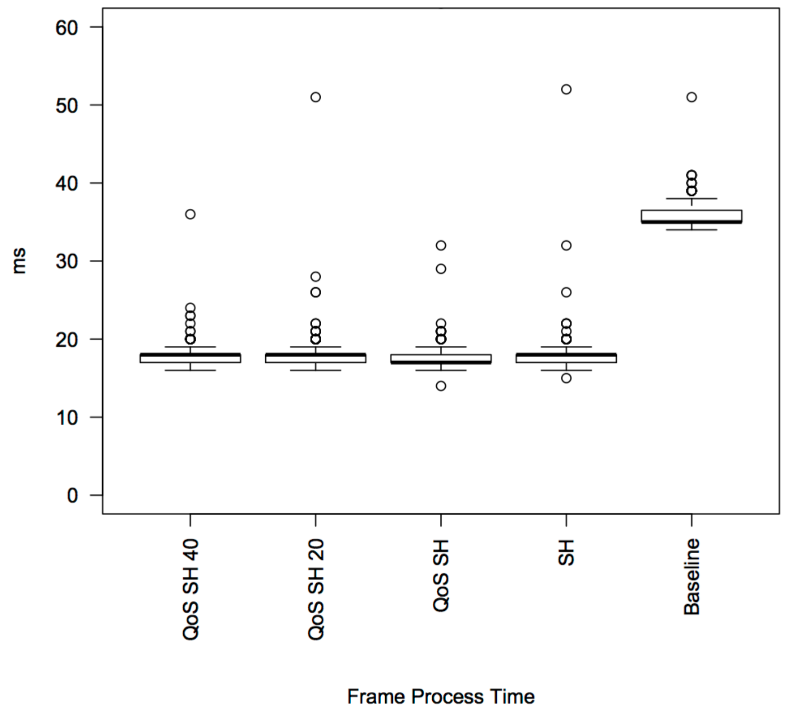


Figure 11

When the number of moving objects was decreased to 1/10, the performance of different setting was very close. The only different was the setting of baseline (QoS disable, Spatial Hashing disable) which did not have any improvement compare with case 1. It was because without the Spatial hashing rendering, the renderer needed to redraw every single object on each frame no matter the object need to be updated or not. The create a lot of unnecessary object redraw.

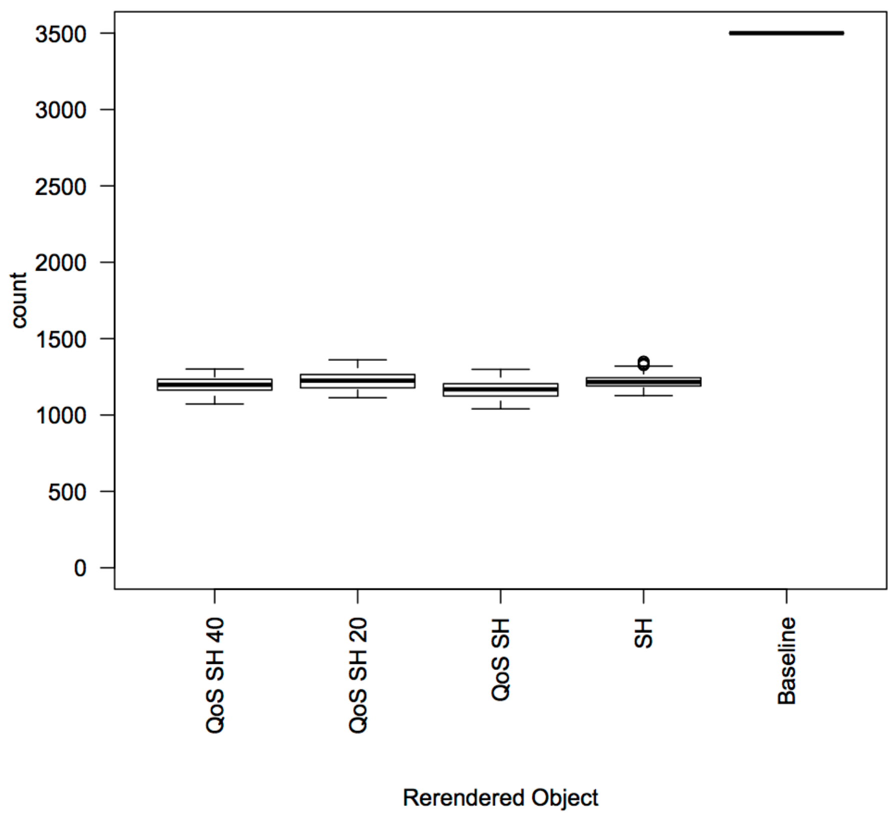


Figure 12

The number of rendered objects is nearly the same, with the Spatial hashing rendering was enabled. Since the computer can process this test case smoothly, there is no any objects was skipped during the rendering because of the QoS.

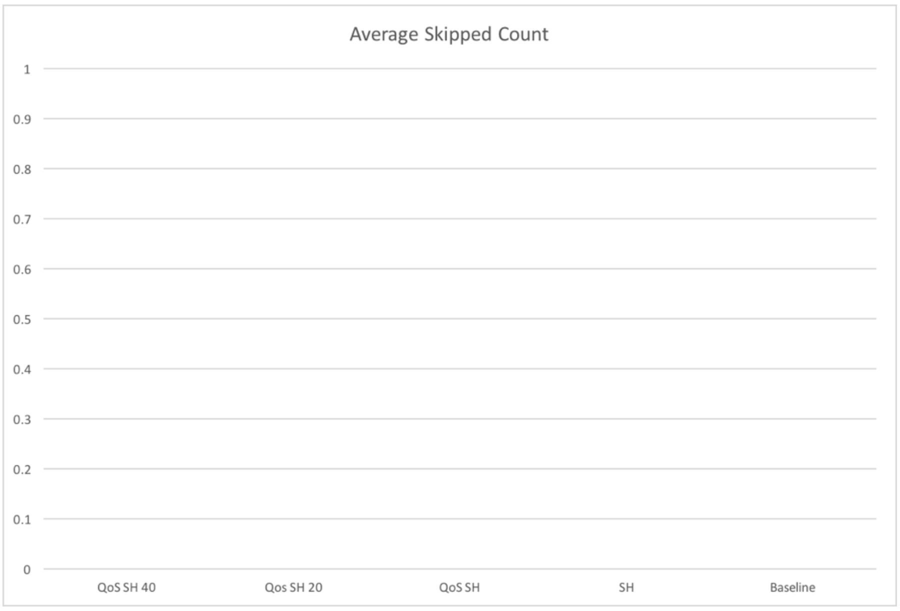


Figure 13

The system processed all objects which is need to be animated and no object was skipped. The QoS system detected the frame rate is above 30fps during the whole animation and keep the QoS Level to allow all objects to be animated.

To conclude this test case, QoS system is not important when the number of the moving objects is very small, but since the overhead of the QoS system is negligible and it is no harm to implement it into the framework.

Further Improvement

Rendering System

The framework currently can support two rendering method, the brute force and the spatial hashing. Each of the rendering method has its advantages and disadvantages. For example, brute force rendering is suitable for rendering the a new scene or the scene that almost every element need to be animated. In these scenarios, brute force rendering will have a better performance because it has no overhead and almost the whole screen need to be redrawn. Spatial rendering can not gain any performance in these cases.

In the other hand, Spatial hashing rendering is good to the scenario which has small portion of the objects need to be animated in the canvas and keep other objects untouched.

The framework can be better if combining these rendering method and choosing the suitable rendering method based on different scenario. But this is a very complicated decision to switching from different rendering method during the run time. Because it has the keep tracking the rendering stage of each object and include the consideration of the QoS. Because when switching from spatial hashing to brute force rendering method it has to redraw all objects in the canvas, this definitely affect the frame rate and the QoS level of the framework.

QoS System

The QoS System can be become much more intelligent if it can detect the QoS level of a object by itself instead of assigned by programmer. But this may very hard to implement and tested in JavaScript. Because JavaScript is a scripting language, it has a lot of limitation in term of data structure, programing flow, logic control, etc.

One of the suggested solution is using the dimension of the objects to determine the QoS level. For example, a larger object has a higher priority to be rendered compare a smaller object. This is feasible because the larger object has higher possibility to draw the attention of the viewer. But some of the small object may be the key element in the scene, such as the cursor. In this case the small object may has the highest priority across all the objects. Maybe the solution can be combine two of the QoS level, the first is the QoS level assigned by user, the second is the QoS level detected by the system using the dimension or position. The first QoS level can override the second one.

In this design, programmer can let the system to control the rendering of the normal objects and manually to force some important objects update and render. This can harvest the most performance and produce acceptable result to viewer.

Collision detection

This feature can be achieved by using the existing spatial hashing mapping. Collision detection is a very essential feature for a game framework. If this framework implemented this feature, it can be more useful and produce a mature demo.

The problem of implement is feature is when the number of objects is excessive, the processing time of check the object is overlapped become unpredictable. Even aid by spatial hashing, the number of calculation still can be numerous.

To make this feasible, let user to set the property of the objects has the collision event or not and keep a collision objects list in each layer. Only the objects inside the list will trigger the collision event. Combining this with the spatial hashing, the calculation time to find the collision objects will become reasonable.

References

Christer, B. (2014, 1 26). *The Shape of Code to Come*. Retrieved 07 11, 2015, from The Shape of Code to Come: http://zufallsgenerator.github.io/2014/01/26/visually-comparing-algorithms/

Lefebvre, S., & Hoppe, H. (2006). Perfect spatial hashing. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH* .

Lu, C., Stankovic, J. A., Son, S. H., & Tao, G. (2002). Feedback Control Read-Time Scheduling: Framework, Modelling, and Algorithms. *Real-Time Systems* , 85-126.

Amadeo, R. (2015). *Google's Dart language on Android aims for Java-free, 120 FPS apps. Ars Technica.* Retrieved June 14, 2015, from Google's Dart language on Android aims for Java-free, 120 FPS apps. Ars Technica.: http://arstechnica.com/gadgets/2015/05/googles-dart-language-on-android-aims-for-java-free-120-fps-apps/

Myth Busting: CSS Animations vs. JavaScript. (2014). Retrieved June 14, 2015, from CSS-Tricks: https://css-tricks.com/myth-busting-css-animations-vs-javascript/

Optimizing JavaScript code. (2015). Retrieved June 14, 2015, from Google Developers: https://developers.google.com/speed/articles/optimizing-javascript

Minimizing browser reflow. (2015). Retrieved June 14, 2015, from Google Developers: https://developers.google.com/speed/articles/reflow

Profiling JavaScript Performance. (2015). Retrieved June 14, 2015, from Google Developers: https://developer.chrome.com/devtools/docs/cpu-profiling

The Basics of Web Workers. (2010). Retrieved June 14, 2015, from HTML5 Rocks: http://www.html5rocks.com/en/tutorials/workers/basics/

Performance Tips for JavaScript in V8. (2012). Retrieved June 14, 2015, from HTML5 Rocks: http://www.html5rocks.com/en/tutorials/speed/v8/

Static Memory JavaScript with Object Pools. (2013). Retrieved June 14, 2015, from HTML5 Rocks: http://www.html5rocks.com/en/tutorials/speed/static-mem-pools/

Improving HTML5 Canvas Performance. (2011). Retrieved June 14, 2015, from HTML5 Rocks: http://www.html5rocks.com/en/tutorials/canvas/performance/

Joe. (2011, August 2). *JavaScript Per-Pixel HTML5 Canvas Image Collision Detection*. Retrieved June 14, 2015, from Play My Code: http://www.playmycode.com/blog/2011/08/javascript-per-pixel-html5-canvas-image-collision-detection/

The secret to silky smooth JavaScript animation! (2012). Retrieved June 14, 2015, from CreativeJS: http://creativejs.com/resources/requestanimationframe/

Flanagan, D. (2011). *JavaScript The Definitive Guide.* O'Reilly.

Zakas, N. C. (2010). *High Performance JavaScript.* O'Reilly.

Monthly logs

FYP Oct 2014 Monthly log

Computer System setup.

Setup working environment.

Setup subversion repository.

Study the properties of JavaScript, such as the typing system.(The typing system of JavaScript is highly affect the performance.)

Reading:

Feature detection and function efficiency within JavaScript

http://davidwalsh.name/feature-detection-function-efficiency-javascript

Typed JavaScript

http://www.2ality.com/2014/10/typed-javascript.html

Native smooth scrolling with JS

http://soledadpenades.com/2014/10/29/native-smooth-scrolling-with-js/

Slides: JavaScript memory management master class

http://addyosmani.com/blog/slides-javascript-memory-management-masterclass/

Improving JavaScript performance by deconstructing the type system.

http://iacoma.cs.uiuc.edu/iacoma-papers/pldi14.pdf

FYP Nov 2014 Monthly log

Working on the interim report I.

Study some import element which highly affect the performance of JavaScript.

Such as the mechanism of garbage collection, memory management.

Decide the code structure of framework.

Reading:

Bailey, C. (2007, September 17). Writing Effective JavaScript Code (or How I Learned to Stop Worrying and Love the Unit Test). Retrieved from Chris' blog: http://chrisbailey.blogs.ilrt.org/2007/09/17/writing-effective-javascript-code-or-how-i-learned-to-stop-worrying-and-love-the-unit-test/

Doyle, J. (2014, January 13). Retrieved from CSS-Tricks: http://css-tricks.com/myth-busting-css- animations-vs-JavaScript/

Doyle, J., & Schoof, C. (n.d.). GreenSock Animation Platform. Retrieved from GreenSock: http://greensock.com/gsap

McAnlis, C. (2013, June 21). Static Memory JavaScript with Object Pools. Retrieved from HTML5Rock: http://www.html5rocks.com/en/tutorials/speed/static-mem-pools/

Rauschmayer, A. (2014, October 28). Statically typed JavaScript via Microsoft TypeScript, Facebook Flow and Google AtScript. Retrieved from 2ality: http://www.2ality.com/2014/10/typed- javascript.html

Wilson, C. (2012, October 11). Performance Tips for JavaScript in V8. Retrieved from HTML5Rock: http://www.html5rocks.com/en/tutorials/speed/v8/

FYP Dec 2014 Monthly log

Meeting with supervisor.

Study the detail approach of how to improve the animation performance.

Which may related to the browser level or virtual machine programming instead of high level programming such as JavaScript. Because of the use of the JS is limited by the browser. But the detail of implementation still need further study.

Reading:

An analysis of the dynamic behaviour of JavaScript programs

http://dl.acm.org/citation.cfm?id=1806598

A Selective Record-Replay and Dynamic Analysis Framework for JavaScript
http://srl.cs.berkeley.edu/~ksen/papers/jalangi.pdf

Managing Performance vs. Accuracy Trade-offs With Loop Perforation

http://people.csail.mit.edu/stelios/papers/fse11.pdf

Better JavaScript animations with requestAnimationFrame

http://www.nczonline.net/blog/2011/05/03/better-javascript-animations-with-requestanimationframe/

Smoothing Slow JavaScript Animation/Parallax

http://www.zachstronaut.com/posts/2009/03/04/smoothing-slow-js-animation-parallax.html

FYP Jan 2015 Monthly log

Trying some different possible of web animation with JavaScript, including SVG, Canvas, CSS3 Animation. Which may become the core element of the framework. Also study how a web browser will repaint the web page (Render Tree, Render Queue).

Reading:

Snap.svg - The JavaScript SVG library for the modern web.

http://snapsvg.io/

SVG.JS - A lightweight library for manipulating and animating SVG.

http://svgjs.com/

Flanagan, D. (2011). Scripted Media and Graphics. In JavaScript: The definitive guide (6th ed.). Cambridge: O'Reilly.

Zakas, N. (2010). DOM Scripting (Repaints and Reflows). In High performance JavaScript. Beijing: O'Reilly/Yahoo! Press.

FYP Feb - April 2015 Monthly log

Finding a suitable approach for an animation framework.

Tried to build different prototype and testing the performance.

Tried different canvas, web worker, browser API.

Started the passing pixel to worker approach in late April , but it is not performance very well.

References:

http://www.playmycode.com/blog/2011/08/javascript-per-pixel-html5-canvas-image-collision-detection/

http://stackoverflow.com/questions/26156477/efficient-sorted-data-structure-in-javascript

http://www.html5gamedevs.com/topic/7735-myths-and-realities-of-canvas-javascript-performance/

http://www.webreference.com/programming/javascript/jkm3/index.html

https://css-tricks.com/myth-busting-css-animations-vs-javascript/

http://www.html5rocks.com/en/tutorials/speed/css-paint-times/

http://www.html5rocks.com/en/tutorials/speed/layers/

http://www.html5rocks.com/en/tutorials/performance/mystery/

http://adambom.github.io/parallel.js/

http://www.sitepoint.com/using-web-workers-to-improve-image-manipulation-performance/

FYP May - April 2015 Monthly log

Since the operation on the DOM element is very expensive in web browser. The framework will mainly use the 2D canvas to achieve the image rendering and animation.

To increase the fps, the main approach is using the "Web Worker" JavaScript API to achieve parallel processing. This can offload the heavy loading job to others thread and keep the main thread (UI thread) responding to user.

I tried different prototype of the framework. One of the prototype is manipulating the low level pixel array directly using the web worker, but since the API "putImageData" is very slow in mainstream web browser. This prototype is hardly to achieve before the API is improved.

Now the focus is try to offload the calculation such as the (locus, coordinate, easing) to the web worker.

Some benchmark for record.

CPU: Snapdragon 800 (4 Cores)

Ram: 2GB

Screen Size: 1080px x 1920px

OS: Android 5.0.2

Browser: Chrome 43.0.23

Rendering 30 Objects 200px x 200px each, random movement

FPS: ~40

HTML5 Rocks - A resource for open web HTML5 developers,. (2012). Performance Tips for JavaScript in V8 - HTML5 Rocks. Retrieved 14 June 2015, from http://www.html5rocks.com/en/tutorials/speed/v8/

HTML5 Rocks - A resource for open web HTML5 developers,. (2013). Static Memory JavaScript with Object Pools - HTML5 Rocks. Retrieved 14 June 2015, from http://www.html5rocks.com/en/tutorials/speed/static-mem-pools/

Play My Code,. (2015). Play, build and share games online!. Retrieved 14 June 2015, from http://www.playmycode.com/blog/2011/08/javascript-per-pixel-html5-canvas-image-collision-detection/

Appendix

Easing function

```
Credit by Gaëtan Renaudeau
Retried from: https://gist.github.com/gre/1650294
EasingFunctions = {
  // no easing, no acceleration
  linear: function (t) { return t },
  // accelerating from zero velocity
  easeInQuad: function (t) { return t*t },
  // decelerating to zero velocity
  easeOutQuad: function (t) { return t*(2-t) },
  // acceleration until halfway, then deceleration
  easeInOutQuad: function (t) { return t<.5 ? 2*t*t : -1+(4-2*t)*t },
  // accelerating from zero velocity
  easeInCubic: function (t) { return t*t*t },
  // decelerating to zero velocity
  easeOutCubic: function (t) { return (--t)*t*t+1 },
  // acceleration until halfway, then deceleration
  easeInOutCubic: function (t) { return t<.5 ? 4*t*t*t : (t-1)*(2*t-2)*(2*t-2)+1 },
  // accelerating from zero velocity
  easeInQuart: function (t) { return t*t*t*t },
  // decelerating to zero velocity
  easeOutQuart: function (t) { return 1-(--t)*t*t*t },
  // acceleration until halfway, then deceleration
  easeInOutQuart: function (t) { return t<.5 ? 8*t*t*t*t : 1-8*(--t)*t*t*t },
  // accelerating from zero velocity
  easeInQuint: function (t) { return t*t*t*t*t },
  // decelerating to zero velocity
  easeOutQuint: function (t) { return 1+(--t)*t*t*t*t },
  // acceleration until halfway, then deceleration
  easeInOutQuint: function (t) { return t<.5 ? 16*t*t*t*t*t : 1+16*(--t)*t*t*t*t }
}
```

requestAnimationFrame polyfill

Credit by Erik Möller

Retried from: https://gist.github.com/paulirish/1579671

```
(function() {
    var lastTime = 0;
    var vendors = ['ms', 'moz', 'webkit', 'o'];
    for(var x = 0; x < vendors.length && !window.requestAnimationFrame; ++x) {
        window.requestAnimationFrame = window[vendors[x]+'RequestAnimationFrame'];
        window.cancelAnimationFrame = window[vendors[x]+'CancelAnimationFrame']
                                   || window[vendors[x]+'CancelRequestAnimationFrame'];
    }

    if (!window.requestAnimationFrame)
        window.requestAnimationFrame = function(callback, element) {
            var currTime = new Date().getTime();
            var timeToCall = Math.max(0, 16 - (currTime - lastTime));
            var id = window.setTimeout(function() { callback(currTime + timeToCall); },
              timeToCall);
            lastTime = currTime + timeToCall;
            return id;
        };

    if (!window.cancelAnimationFrame)
        window.cancelAnimationFrame = function(id) {
            clearTimeout(id);
        };
}());
```

Web Worker polyfill

Credit by Timothy Chien

```
https://github.com/timdream/simworker
```

```
"use stricts";

if (!window.Worker || window.forceIframeWorker) {
        if (window.Worker) window.nativeWorker = window.Worker;
        window.Worker = function (script) {
                var worker = this;

                // prepare and inject iframe
                worker._iframeEl = document.createElement('iframe');
                worker._iframeEl.style.visibility = 'hidden';
                worker._iframeEl.style.width = '1px';
                worker._iframeEl.style.height = '1px';
                worker._iframeEl.onload = worker._iframeEl.onreadystatechange = function () {
                        if (this.readyState && this.readyState !== "loaded" && this.readyState !==
"complete") return;
                        worker._iframeEl.onload = worker._iframeEl.onreadystatechange = null;
                        var w = this.contentWindow,
                        doc = this.contentWindow.document;

                        function injectScript(script, callback) {
                                var scriptEl = doc.createElement('script');
                                scriptEl.src = script;
                                scriptEl.type = 'text/javascript';
                                scriptEl.onload = scriptEl.onreadystatechange = function () {
                                        if (scriptEl.readyState && scriptEl.readyState !== "loaded"
&& scriptEl.readyState !== "complete") return;
                                        scriptEl.onload = scriptEl.onreadystatechange = null;
                                        doc.body.removeChild(scriptEl);
                                        scriptEl = null;
                                        if (callback) {
                                                callback();
                                        }
                                };
                                doc.body.appendChild(scriptEl);
                        }

                        // Some interfaces within the Worker scope.

                        w.Worker = window.Worker; // yes, worker could spawn another worker!
                        w.onmessage = function (ev) {}; // placeholder function
                        var postMessage = function (data) {
                                if (typeof worker.onmessage === 'function') {
                                        worker.onmessage.call(
                                                worker,
                                                {
                                                        currentTarget: worker,
                                                        timeStamp: (new Date()).getTime(),
                                                        srcElement: worker,
                                                        target: worker,
                                                        data: data
                                                }
                                        );
                                }
                        };
                        w.postMessage = w.workerPostMessage = postMessage;
                        if (w.postMessage !== postMessage) {
                                // IE doesn't allow overwriting postMessage
                        }
                        w.close = function () {
                                worker.terminate();
                        };
                        w.importScripts = function () {
                                for (var i = 0; i < arguments.length; i++) {
                                        injectScript(window.Worker.baseURI + arguments[i]);
                                }
                        }

                        // inject worker script into iframe
                        injectScript(window.Worker.baseURI + script, function () {
                                worker._quere.push = function (callback) {
                                        if (!worker._unloaded) {
                                                callback();
                                        }
                                };
                                if (!worker._unloaded) {
                                        while (worker._quere.length) {
                                                (worker._quere.shift())();
                                        }
                                }
                        });
                };
                this._iframeEl.src = window.Worker.iframeURI;
                (document.getElementsByTagName('head')[0] ||
document.body).appendChild(this._iframeEl);

                worker._quere = [];
                worker._unloaded = false;
        };
        window.Worker.prototype.postMessage = function (obj) {
                var worker = this;
                setTimeout(
                        function () {
                                worker._quere.push(
                                        function () {
                                                // IE8 throws an error if we call
worker._iframeEl.contentWindow.onmessage() directly
                                                var win = worker._iframeEl.contentWindow, onmessage =
win.onmessage;
                                                onmessage.call(win, {data:obj});
                                        }
                                );
                        },
                        0
                );
        };
        window.Worker.prototype.terminate = function () {
                if (!this._unloaded) {
                        (document.getElementsByTagName('head')[0] ||
document.body).removeChild(this._iframeEl);
                }
                this._iframeEl = null;
                this._unloaded = true;
        };
        window.Worker.prototype.addEventListener = function () {
        };
        window.Worker.prototype.removeEventListener = function () {
        };

        window.Worker.notNative = true;
        window.Worker.iframeURI = './worker.iframe.html';
        window.Worker.baseURI = '';
}
```

Basic API of the Framework

Initialize the framework with an empty dom element such as a <div> tag.

```
Var fw = new Framework(“domID”);
```

Adding a new layer

```
Var layer = fw.createLayer(“layerName”);
```

Adding a model

```
var model = layer.addModel({
        type:”image”,
        url:”img/pic.png”,
        x: 100,
        y: 200
});
```

Animate a model

```
model.addAnimation({
        x: 200,
        y:300,
        duration: 100,
        easing: “linear”
});
```

Set zIndex of model

```
model.setZIndex(1);
```

Remove a model

```
layer.removeModel(“modelName”);
```

Remove a layer

```
fw.deleteLayer(“layerName”);
```