Specialist Techniques for Graphics and Animation

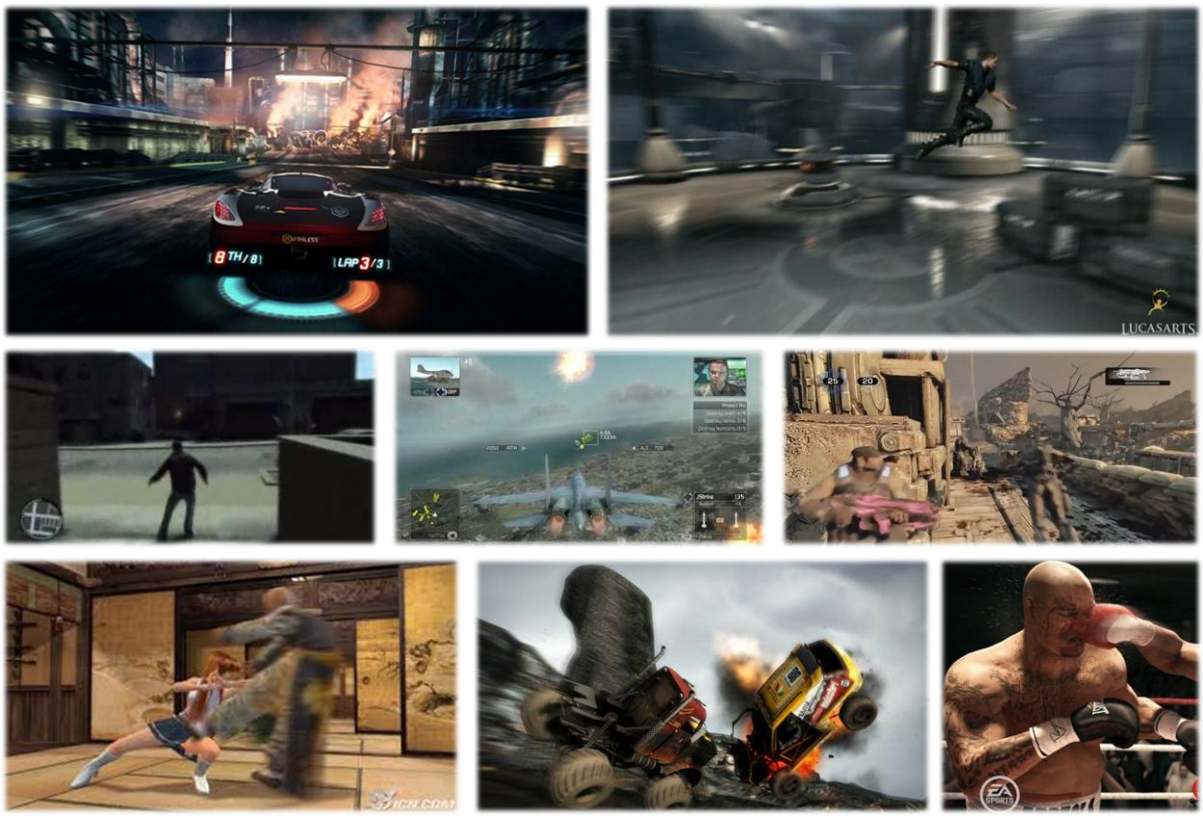
Assignment One

Thomas Sampson – MComp Games Software Development

1. Introduction

This report will focus on the art and implementation of motion blur in real time applications, specifically applied in the context of video games across a variety of gaming genres. Motion blur is becoming increasingly prevalent across the current generation of games on the market and can be tailored to suit many different scenarios based on the requirements of the artist and art direction of a particular title. When motion blur is applied to a particular scene (or group of objects within that scene) it can create a great sense of **speed**, **thrill**, **movement**, **dizziness**, **delirium and intensity** [1], all of which can add greatly to the **realism** and **immersion** provided by the game. In addition to reviewing the technical approaches to implementing the motion blur effect, I will also discuss how the effect is best configured, and what parameters are required to provide artistic control and manipulation of the motion blur processing algorithm.

Motion blur effects have been commonplace in graphics manipulation packages (Adobe® Photoshop currently provides eleven unique and highly customisable blurring filters) for many years and their widespread use can be seen across a range of media such as; magazines, advertisements, websites and cinematic special effects. Motion blur can even be achieved easily with traditional and digital photography by increasing the period of exposure by manipulating the cameras shutter speed. Although motion blur effects naturally lend themselves to fast paced racing games, the collection of images below show just how versatile the technique can be across a multitude of gaming genres.



From top left to bottom right: EA - **Split Second**, Lucasarts – **Force Unleashed II**, Rockstar Games **Grand Theft Auto IV** (Drunk Effect), Ubisoft - **Tom Clancy Hawx II**, Epic games - **Gears of War II**, Team Ninja - **Dead or Alive 2**, Evolution Studios - **Motorstorm Pacific Rift**, EA Sports – **Fight Night**

[1] These different side effects of motion blur would rarely be used in conjunction, but rather in isolation to fit a particular genre or style of gameplay.

2. Review of Existing Solutions and Literature

Before attempting to implement any kind of motion blur effect I first reviewed articles from different sources to gain a broader understanding of the technique. There are many different ways to implement motion blur in real-time applications, each with their own strengths and weaknesses. In this section I will review each potential approach and considering the merits of each, choose a suitable candidate for implementation.

Motion Blur Geometry

This technique involves the calculation of additional geometry, appended to an object within the scene to simulate the presence of motion blur. The additional geometry is generated by creating additional polygons before and behind the moving object, along its edges. The alpha value of the generated vertices is interpolated such that the newly formed polygons are opaque where they meet the edge of the original object, and fully transparent towards their outer edge.

Figure 1 - Microsoft DirectX SDK "*Motion Blur 10*" Sample demonstrates motion blur geometry.

This technique is perfectly reasonable in some scenarios and produces a believable result as demonstrated in Figure 1. However the main issue is that the computation of additional geometry can be expensive as it must be first generated on the CPU, then placed into a buffer which is transferred to the GPU each frame, only to be evicted from GPU memory on the next frame as new blurring geometry is calculated. This can vastly increase the vertex bandwidth (especially when the scene contains many moving objects) to the GPU and ultimately requires more *draw prim* calls each entailing additional alpha calculations. Most new GPUs support a new feature called "Geometry Shading"; allowing new geometry to be created directly on the GPU, pushing it straight into the rasterizer and bypassing the CPU->GPU bottleneck entirely. Unfortunately, for the purpose of this report only DirectX 9 support is available and Geometry Shading requires DirectX 10 or greater.

Motion Blur via the Accumulation Buffer

The Accumulation Buffer is an additional image buffer that is used to accumulate composite images. The Accumulation Buffer naturally lends its self to motion blur as moving objects are rendered multiple times along their path of motion and then the images averaged to create the illusion of blur (as demonstrated in Figure 2). The main drawback of using the Accumulation Buffer is that it is expensive and can prevent interactive frame rates. However, the Accumulation Buffer approach closely resembles the motion blur we are conditioned to expect from movies (which was originally an artefact of the projection equipment in cinemas) and therefore can produce excellent reference images on which to judge the realism of other less computationally expensive motion blurring techniques.

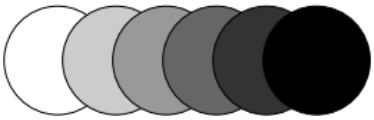


Figure 3

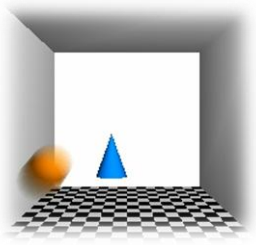


Figure 2 – Open GL SDK Sample demonstrates use of Accumulation Buffer to create a motion blur effect.

Velocity Buffers

Velocity buffers are used to store a screen space (two dimensional) velocity per-pixel for the entire screen. These screen space velocity values are calculated as follows:

1. For each frame, store a previous and current world transform for each object within the scene
2. For each vertex comprising an object, calculate its previous and current position in world space (using the previous and current world transform matrices) producing a three dimensional vector with both direction and magnitude (where the magnitude represents the distance travelled by that particular vertex between the previous and current frame)
3. Interpolate the per vertex direction vectors across each fragment of the polygon
4. Project the three dimensional direction vectors at each fragment to relative screen-space (two dimensional) co-ordinates

The result is a velocity buffer which contains an array of two dimensional screen space co-ordinates / directions. One neat thing about velocity buffers is that they map directly to the back buffer and can be visualised by mapping the x and y values of the direction vector at each pixel, to the desired channels (r, g, b, a) of a texture. This is demonstrated in Figure 4.

Figure 4 – Killzone 2 Velocity map values mapped to red and green channels of texture.

Velocity buffers are usually written to an off screen surface (which can be computationally and memory expensive) and do not affect the initial render pass of the scene. However, once the scene has been rendered in its entirety (without motion blur) a post processing effect can be used to blur the original render by sampling values from the velocity map.

For each pixel in the original image, the corresponding screen space direction can be sampled from the velocity map. Samples can then be taken from the neighbouring pixels of the original image along this direction and averaged to compose the final pixel colour. The number of samples taken will directly influence the amount of motion blur applied to the final image.

Radial Blur

Figure 5 – Assassins Creed II uses radial blur to represent fast paced forward movement of the protagonist.

Radial blur is an effect used in many racing games and in some third person action games (as shown in Figure 5) to create the illusion of forward motion and to focus the attention on a particular item (usually the players car or character). Some cut scenes also employ this technique to represent events that do not occur in the real world, such as a dream or a character's thoughts or epiphanies. This effect is created by blurring each pixel by sampling along the direction given by the pixels position in relation to the screen centre. The number of samples taken is usually proportional to the pixels distance from the screen centre.

Motion Blur as a Pure Post Processing Effect

This approach is an adaptation of the velocity buffer approach which provides motion blur purely as a post processing approach without reliance upon a separate velocity buffer. The way this approach works is to compute the velocity buffer "on the fly" during a post processing pass; by extracting the world space position of each vertex from the depth buffer and then using the cameras previous and current *view-projection* matrix to calculate the direction/distance travelled (based on its movement in relation to the camera). Once this value is obtained it can then be mapped to screen space and used to blur the original image exactly as demonstrated in the previous section.

This approach is fairly new and has several benefits over the aforementioned velocity buffer approach. Firstly, this technique does not require the generation of a separate velocity buffer which can be both time consuming, and doubles the texture memory footprint of the rendering system. Reducing the amount of computation and memory consumption makes the effect better suited to implementation on older less powerful consoles with tighter memory constraints. This makes this effect much more appealing to developers working on cross platform titles, allowing them to reach a bigger target audience and potentially benefit from sales across multiple formats.

Equally importantly, this technique requires zero modification to the current rendering system. All values are computed during the frames post processing stage and are extracted from the depth buffer (which in 99% of cases is already being automatically populated by the GPU). The fact that the original vertex and pixel shaders require no special modification makes this technique perfectly suited to being retrofitted into existing game titles with minimal effort. As long as the current title has a large enough slice of frame time to spare, this technique can be bolted onto the end of the rendering pipeline in isolation (as shown in Figure 6), without affecting the games frame rate.

One drawback of this approach is that it does not take into account the relative movement of objects about the scene, only in relation to the cameras position and angle. However as the ultimate goal of this report is to apply motion blur to a first person exploration game, where blurring is a result of the movement and behaviour of the player (hence the camera), this problem can be overlooked.

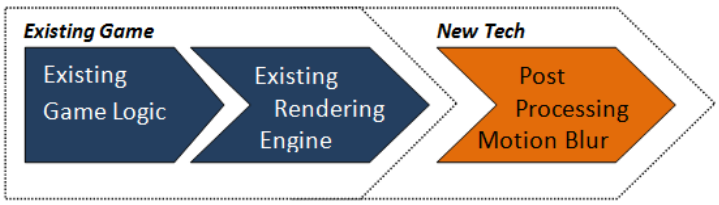


Figure 6

3. Design

This section describes how I can set up and configure an environment which will act as a test bed for implementing motion blur.

Scene Setup

The scene I intend to use to demonstrate motion blurring will comprise of a single car driving upon a road at high speed. The fast moving car will make a good candidate for applying motion blur techniques, and fits the context of the experiment. I will also animate the car slightly to show it bouncing slightly to simulate the rough surface of the road.

Controlling Effect Parameters

I plan to be able to put the simulation into