UNIVERSITY OF WATERLOO
Faculty of Physics & Astronomy

ACOUSTIC MODELLING USING MEL-FREQUENCY CEPSTRAL COEFFICIENTS

Sysomos
Toronto, Ontario

Prepared by

Thomas C. Fraser
3A Mathematical Physics
ID 20460785
January 10, 2016

154 Quarry Ave.
Renfrew, Ontario
K7V 2W4

January 10, 2016

Mr. Jeff Chen, Department Chair
Department of Physics and Astronomy
University of Waterloo
200 University Avenue West
Waterloo, Ontario
N2L 3G1

Dear Mr. Chen:

I have prepared the enclosed report "Acoustic Modelling Using Mel-Frequency Cepstral Coefficients" as my 3A Work Report for my work term spent at Sysomos in Toronto, Ontario. This is my second work term report. The purpose of this report is to examine the problems associated with a popular technique used in application design known as a sprite sheet. I aim to convince anyone familiar with the ideas discussed to consider the solution I propose to solve these problems. This report uses the techniques I developed while tactling these problems but is targeted at a audience with a wider range of applications.

Sysomos is currently working on a multi-platform video game. My supervisor, Elyot Grant, assigned me with overcoming some limitations of graphics API used.

This report was written entirely by me and has not received any previous academic credit at this or any other institution. I give permission to Sysomos to keep a copy of this report on file and use it as necessary in the future.

Sincerely,

Thomas Fraser

Thomas C. Fraser
ID 20460785

Table of Contents

List of Tables and Figures

Summary

1.0 Introduction

2.0 The Problem

One of the primary objectives of a graphics or user interface developer is to optimize as many aspects as possible in order to provide and pleasant experience for the user. Umoung many of these objectives is having a well designed visual hierarchy, attractive graphics and theme, and a smooth experience. Developing a smooth performance is tricky, especially when developing applications with dynamically changing resolutions, or one with many assets, a complicated render-order, or that displays images that can be modified by the user.

2.1 Reducing Draw Calls

In order to improve the end-user experience, many parameters need to be optimized. Some of these paramters include the number of frames per second (FPS), the total memory used by the application, the CPU load, the number of draw calls, etc. All of these parameters are intimately connected, but one of the most important factors that affects performance is draw calls. Draw calls, loosely speaking, represent the number of distinct rendering states the GPU transitions through while rendering an given frame. It is a quantity that can change anytime during runtime.

2.2 Render States

Defining a unique render state is complicated. It can depend on the GPU hardware, the way an image is blended into the images behind it, the transparency of the image, the source texture object or the smoothing values associated with the texture. However, for the following analysis, we will simplify the notion of a render state to be the be associated directly with the concrete texture being rendered from. Therefore, a draw call corresponds to switching from one concrete texture to another.

2.3 Render Order

The GPU renders any given frame within the application by iterating though each of the images that might appear on the screen, calculating the resultant position on the screen based on a global matrix associated with the image that encodes it's position, scale, skew, and orientation. The order in which the images are rendered directly affects the number of render states. The worst case scenario occurs when every image on the screen is rendered from it's own distinct concrete texture. As a result, when the GPU switches from rendering one image to the next, it needs to switch which concrete texture is has active, increasing the number of draw calls by one. With potentially thousands of images on the screen, the GPU would have to make 1000 draw calls. This is extremely inefficient and can be improved by combining the textures of each image together into one larger image that shares the same concrete texture.

3.0 Conclusions

Sprite Sheets limit the Features and Development of the Application

Traditional sprite sheets are limited by the size of sprite sheet the GPU can handle. This indirectly encourages developers to only support specific resolutions, limit user control over graphics, and to reduce the visual complexity of the application to solve this problem.

Dynamic sprite sheets allow for a more transparent development enviroment.

Dynamic sprite sheets are implemented such that the details and location of individual textures within a dynamic sprite sheet are keep unknown. Texture retrieval is simplified and texture allocation is more controllable.

Dynamic sprite sheets reduce server costs.

Streaming assets individually from a host server to a user application allows for intelligent caching of assets. It means updates to the code-base do not enforce the end-user to re-download assets. Effectively, each image is only downloaded once, minimizing load on the server and reducing bandwidth costs.

4.0 Recommendations

Consider streaming all necessary assets from an external server.

With dynamic sprite sheets, it is a good practice to store every graphic on an external server that can be downloaded by the application only once and cached. These individual files can be allocated to a dynamic sprite sheet only when needed, reducing memory, and improving performancing.

A dynamic atlas implementation should be used for any application already using sprite sheets.

All developers that currently is used traditional sprite sheets should port their systems to use dyanmic sprite sheets. Dynamic sprite sheets are strictly better for any application where it is appropriate. They accomplish all of the same tasks while removing limitations such as multi-resolution support and the complexity associated managing many traditional sprite sheets.

Bitmap fonts need to be pre-rendered at runtime.

Dynamic sprite sheets offer a very unique implementation for combining the clarity of vector fonts with the performance of bitmap fonts. It is recommended that all fonts used by an application are vector fonts rendered as bitmap fonts at runtime.

References

1 Williams, M. (2013). 10 Great Full Game Sprite Sheets From GraphicRiver. Retrieved 11 May, 2015 from http://gamedevelopment.tutsplus.com/articles/10-great-full-game-sprite-sheets-from-graphicriver–gamedev-3797

2 CodeAndWeb GmbH. (2015). TexturePacker online documentation. Retrieved 11 May, 2015 from https://www.codeandweb.com/texturepacker/documentation

3 Unity Technologies. (2015). Unit Documentation: Draw Call Batching. Retrieved 11 May, 2015 from http://docs.unity3d.com/Manual/DrawCallBatching.htmldocumentation

4 Champion, R., Paci, T. & Vardon, J. (2012). PD 2: Critical Reflection and Report Writing. Retrieved 1 March, 2012 from https://learn.uwaterloo.ca/d2l/le/content/80224/viewContent/605550/View

Note: [4] was referenced to format this report.

Glossary

Concrete Texture A texture that is a source texture. It's source data comes from a bitmap object. 2, 3

Image The displayobject the GPU renders to each frame. Not to be confused with a bitmap. 2, 3, 5

Sprite Sheet Sprite Sheets are the tradtional technology used to minimize the number of draw calls. A collection of smaller image files compiled into one large image by third-party software. i, 5

Texture A texture stores the information that represents an image to be displayed. The bulk of the texture data is stored on the system GPU. 2, 5