
Report TNCG14 - Advanced Computer Graphics

Anders Nord - andno922@student.liu.se

June 8, 2013

1 INTRODUCTION

When a DJ is playing there is almost always a visual show going on around the stage. This can be done in a million ways and is often pre-made or created by a separate person called a Vjay. The Vjay is doing the visual show live to the beat and feel of the music. A visual show can contain lights in different colors, blinking lights(strobe), projections of basically anything from static pictures switching, movie-clips being played in pace to the music or computer made graphics.

The goal with the project was to create a program for Djs that lets them be able to manage this show simultaneously to DJaying. The main idea is that the person DJaying shall be able to interact with the visual show whenever it is desired, and when it is not, it should act on its own based on conditions set earlier by the DJay.

To fully understand the content in this report, a basic understanding of audio-analysis and OpenGL with the use of shaders is preferred.

2 METHOD

This project has been implemented in programming language C/C++. API's and librarys that has been used are the audio library BASS¹, the math library glm² and OpenGL combined with GLSL. The API GLFW³ was used for easy access to create an OpenGL window and a few other things.

First there will be a light description and explanation of what was done. Then a more specific explanation of how the visualization was done will be presented.

In theory what needs to be done:

1. Stream data from a musicfile
2. Transform the data into the frequency domain
3. Use the frequency-analysis to connect variables to different frequencies.

¹<http://www.un4seen.com/>

²<http://glm.g-truc.net/>

³<http://www.glfw.org/>

4. Visualize this data in real-time
5. Create interactivity for the user

What was done:

2.0.1 STREAM DATA FROM A MUSICFILE

What is needed here is realtime datasamples from a live stream of music. To achieve this the BASS-library was used. BASS library is C++ compatible and gives access to a live audio stream. The BASS library can get the sample data from the audiostream.

2.0.2 TRANSFORM THE DATA INTO THE FREQUENCY DOMAIN

To be able to make any good use of this data, it needs to be transformed into the frequency domain. This can be managed by using the Fast Fourier Transform. The BASS library gives you the possibility to choose if you want the data to be transformed by an FFT and at the same time, if desired, apply a Hann window to the sample data to reduce leakage. The most accurate FFT in BASS-library returns 8192 transformed samples.

2.0.3 CONNECT VARIABLES TO DIFFERENT FREQUENCIES

To connect different variables to a correct interval of frequencies, the sample-rate must be known. The sample-rate is used to calculate the corresponding frequency from a 8192 sample-interval that is obtained from the BASS FFT. The desirable thing would be to connect something to the three main-frequency-intervals. This is something that gives the visual show a continuity.

2.0.4 VISUALIZE THIS DATA IN REAL-TIME

Since the data needs to be processed in real time and synced with the music, the natural choice is to use the GPU to draw the visuals. OpenGL gives low-level access to the graphics card and to get direct access to the GPU, shaders with GLSL is used.

2.0.5 CREATE INTERACTIVITY FOR THE USER

To give the user an interactive experience there is a lot of things one can do. Turn booleans on and off, increase and decrease values and create different layers with shaders. By combining these things a lot of different effects can be achieved.

2.0.6 A MORE TECHNICAL WALK THROUGH OF THE VISUALIZATION-PROCESS

First the BASS FFT is performed on the CPU. Then these samples are used to create a simple EQ-bar as seen in figure 2.1 to see that the data-samples are gathered correctly.

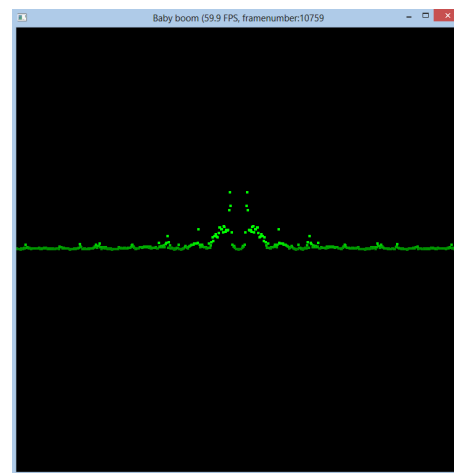


Figure 2.1: A first equalizer with bassfrequencies in the middle

To be able to create something more interesting, a sphere is created out of a triangle-list attached to a VAO(Vertex Array Object). This is done by a function written in C-code. By doing this way, the option how to draw this list later is presented. The different choices implemented are `GL_TRIANGLES`, `GL_QUAD_STRIP`, `GL_TRIANGLE_FAN` and `GL_POINTS`.

To give an example, in figure 2.2 the `GL_TRIANGLES` are chosen while in figure 2.3, `GL_TRIANGLE_FAN` is chosen.

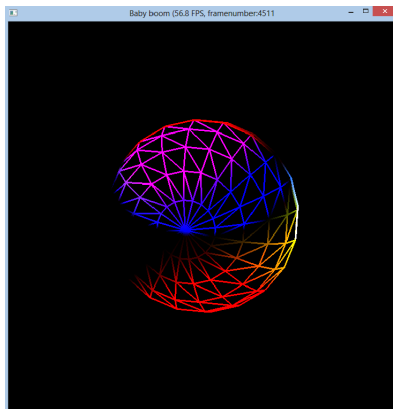


Figure 2.2: The sphere drawn using `GL_TRIANGLES`

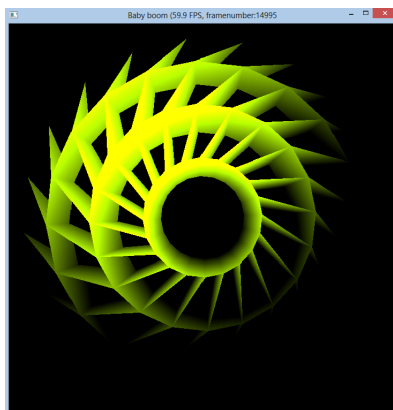


Figure 2.3: The sphere drawn using `GL_TRIANGLE_FAN`

This simple trick creates a rather stunning visual difference. Another good thing with this way of creating a triangle-list is that a choice of how many triangles that the sphere should consist of exists. This creates different levels of detail and can also give big visual differences.

To optimize the performance all of the transformation matrices were multiplied with each other on the CPU with glm and sent into the shader as one final matrix. This is because the shader would have to recalculate all of the transformation matrices for each vertex or pixel.

To create a bigger variety on how the spheres are displayed, there are three ways on how to set the color implemented. When the sphere is sent to the shader a choice of letting the set color be multiplied by the normal, affected by a light that is circling around the scene, affected by both the normal and the light or just be the set color is presented.

If the light is activated there will be pixel shading as can be seen in figure 2.3 where the light is in the upper left corner. In figure 2.2 the sphere is colored by both the light (above the sphere) and the normal.

To create effects that changes the whole structure of the resulting image, three final shaders are created. This means that one of these three shaders will draw to the renderbuffer which will then appear on the screen.

To be able to achieve this, an FBO (Frame Buffer Object) is created and gets a texture and depthbuffer attached to it. So when the spheres are being drawn the final texture is saved into the texture attached to the FBO instead of the being sent to the screen. The final shaders can now use this texture of everything that has been sent to the GPU and also use the depthbuffer.

So the three implemented final shaders are

a blur-shader(see figure 2.4), a depth-buffer-shader(see figure 2.5) and a shader that just shows the texture as it is.

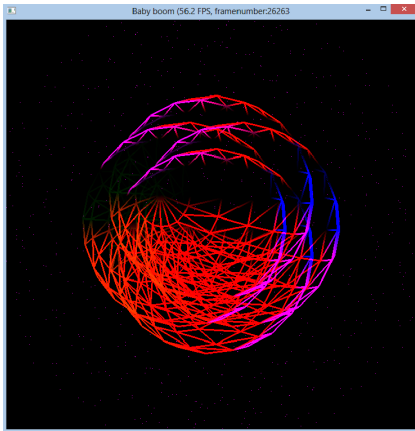


Figure 2.4: The texture drawn using the blur-shader with a high kernel-value

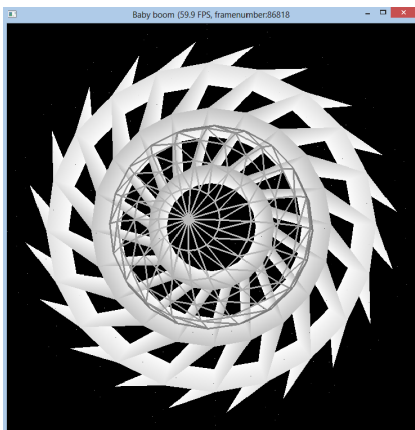


Figure 2.5: The texture drawn using the depthbuffer

To make the interactivity even better the DJay can turn on and off the different final shaders and for example change the kernel value live in the blur-shader.

Other implemented things that can be affected:

- Intensity of the color - This can be de-

termined by the value of the frequency analysis. This is the case in figure 2.3.

- Strobe - you can strobe every part individually and the whole screen as well.
- Resolution - The resolution can be chosen and needs to be equilateral (dimension * dimension)

3 RESULTS

The result of this project is a program that can stream musicfiles in the formats MP3, MP2, MP1, OGG, WAV and AIFF and visualize it in a graphical form.

These are a few pictures that show the results and some different effects that can be achieved.

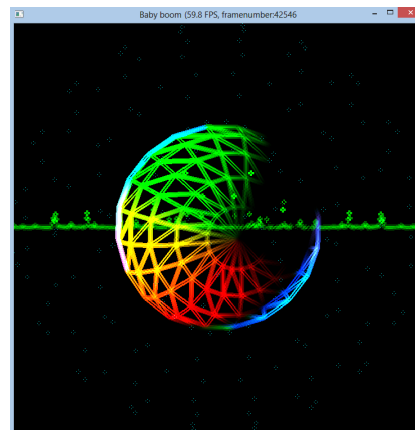


Figure 3.1: The texture drawn using the blur-shader with a low kernel value

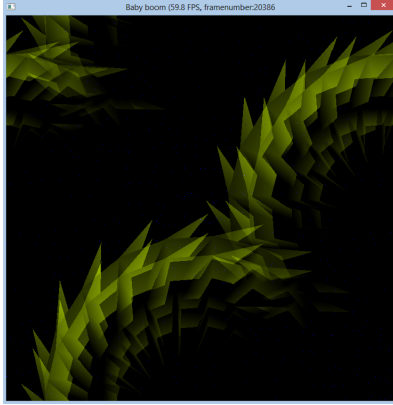


Figure 3.2: The texture drawn using the blur-shader with a high kernel value

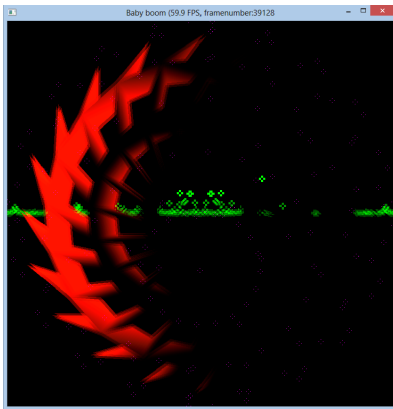


Figure 3.3: The texture drawn using the blur-shader with a low kernel value

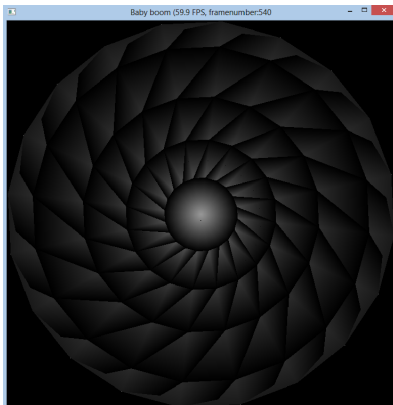


Figure 3.4: The texture drawn using the depth-buffer while also using the strobe-function

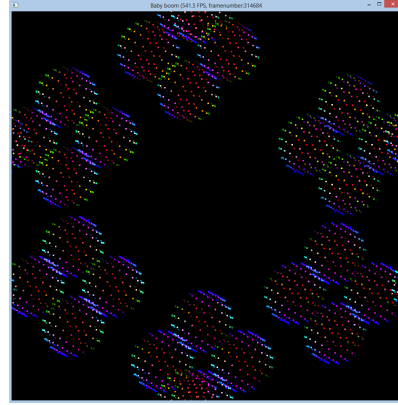


Figure 3.5: The texture drawn using the blur-shader, 541.5 FPS, resolution: 1024 * 1024

4 PERFORMANCE

4.0.7 COMPUTER SPECS

These test were running on this machine:

Processor: Intel Core i7 3610QM 2,3 GHz

Memory : 8 GB of DDR3 1600 MHz SDRAM

Graphics card: Intel(R) HD Graphics 4000 that has a graphics memory of 1792 MB

4.0.8 PERFORMANCE MEASUREMENTS

Table 4.1: Performance results in FPS when the application has every effect turned on, blur-shader as final shader and while strobing

Resolution	FPS
128 * 128	1380
512 * 512	1250
1024 * 1024	350
1280 * 720	400
1920 * 1080	160

These are the test-results when the program is maxed out and shows the lowest FPS noted for that particular resolution.

Any similar program has been hard to find so a comparison has not been possible.

In the future there might be a way to parallelize the FFT on the GPU for better performance.

A limitation right now is that only equilateral resolutions look good.

5 FUTURE WORK

For future work there are a few main things that could be implemented:

- Read the audio stream from the main audio output
- Analyze the sound and find out the song's BPM(Beats Per Minute). The BPM could be used for a lot of different things, especially when the visuals should create patterns by itself
- Create a GUI/interface in a separate window for the user to more easily be able to understand and use the program
- parallelize the FFT on the CPU to get better speed (might be needed when the visuals get more advanced)
- Combine the final shaders in different ways. Let them blend together or other things
- Be able to let the user activate a "picture-mode" which will make the program switch between hand-picked pictures to the rhythm of the song
- Be able to stop the music, change volume and change track

6 CONCLUSION AND DISCUSSION

To summarize this project the theory was consistent with what had to be done.

It was hard to find a good audio library that could do what was desirable. Both in speed and simplicity.

It is relatively easy to get the bass-frequencies to show in the visuals but the middle and top-frequencies are harder to distinguish. There are many ways the frequency value can be calculated. Two easy examples are either to use the highest value in a chosen interval or calculate the mean value.

It is hard to get a lot of variety in the visuals when everything has to be built from scratch. A good solution might have been to make a plug-in to a 3D-program. Then the FFT-values could have easily been connected to the parameters in there. This would also have given a lot of other possibilities such as render videos to the music and so on.

A rather big limitation is that everything must happen live in real-time. Because of this you cannot be sloppy when choosing how to show your visuals and you need to think twice about how demanding they are in processing power.

The streaming of data-samples gives the positive effect that if the visuals lose frame-rate because of demanding tasks, then they still won't fall behind the music.

This has been a really rewarding project and has given insight into many different parts of audio-knowledge and graphical applications.