VoRMD

Volume rendering on mobile devices.

Manuel Hochmayr, BSc
e0627715@student.tuwien.ac.at

ABSTRACT

Nowadays volume rendering is a very important topic. There are many programs for PC that render volumes very fast and display all kind of information the user is interested in. However portable devices have also become very popular these days, but there are almost no real-time volume render programs for them. Due to hardware improvements in the mobile sector, e-mail, browsing and games have gone portable- so now real time volume rendering.

VoRMD is a project about volume rendering on a mobile device. It was planed to do a comparison between OpenGL ES 2.0 and WebGL and their performances on doing a volume rendering on a mobile device. Because of the lack of support of WebGL on mobile web browsers, the WebGL part was dropped. The new goal was implementing a volume-rendering framework in OpenGL ES 2.0 with GLSL for an Andorid 2.3 device. The program supports Android 2.3 devices having a GPU chip with 3D texture extension. It is possible to load volumes in ".dat" format and transfer function images in ".png "format. It uses 4 different kinds of shader, MIP, averaging, composition and XYZ slicing, for calculating and displaying a volume file. All that is done in real time on the device. The user can use touch input to rotate and scale the volume. Furthermore the program adapts the sampling rate according to the frame rate, to allow better interaction when transforming the volume.

GOAL:

The original goal of the project was to implement a simple volume renderer on a mobile phone with Android 2.3. It should only use a MIP (maximum intensity projection) shader to do a rendering with OpenGL ES 2.0 and WebGL. WebGL was somehow working on the device by downloading *FireFox* mobile. However it was not possible to integrate the *FireFox* mobile window into the app and there was no texture 3D support in WebGL. Therefore the original goal was dropped and a new one was formulated which was now implementing a framework for volume rendering in OpenGL ES 2.0. The framework should support 4 different kinds of shaders MIP, Averaging, Compositing and Slicing in X Y and Z direction. Furthermore loading different volumes and transfer functions and adaptive sampling as well should be possible.

TECHNOLOGY:

VoRMD targets an *Android 2.3* device. For developing the program *Eclipse* with *Android SDK 10*, for the Java part and *NDK r6* were used. The user interface uses standard Android interface components. An overlay is used to display the UI which is not the best solution but acceptable. Furthermore for development a *HTC Desire HD* with *Adreno 205* GPU was used. The GPU supports the texture 3D extension of OpenGL ES 2.0.

.dat fileformat:

The first 6 bytes are the header of the file. The first two bytes are the width, the next two are the height and the last two bytes are the depth of the volume. The rest of the file describes the volume. A data set has the size of 16 bytes, however only 12 of them are used. For further information visit:

https://www.cg.tuwien.ac.at/courses/Visualisierung/Angaben/Bsp1.html

PROGRAM

Start

You can open the program by simply clicking the VoRMD icon on your Android menu screen. The program will start and a standard volume will be loaded – lobster.dat. In the middle of the screen you can see the actual volume. On the left hand side top corner an orientation cube is visible. The cube helps orientating when rotating and scaling the volume. Pressing the *dz* plus and *dz* minus buttons on the left and right hand side corner can change the sampling rate.

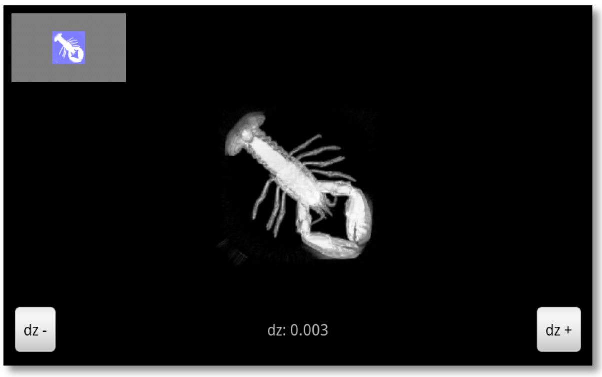

Figure 1: Start screen

The menu

Pressing the Menu button on your mobile device opens the menu. It will show up at the bottom of the screen. There are 4 different options to select. Pressing Load… the load menu opens. Via this menu volumes and transfer functions can be loaded by simply finding them by a separate file chooser window. To load own volumes one have to simply put them on a SD-Card of the mobile phone. This can be done for the transfer function as well. The next option in the menu is shader. By clicking it a window with all 4 different kinds of shaders to choose appear.

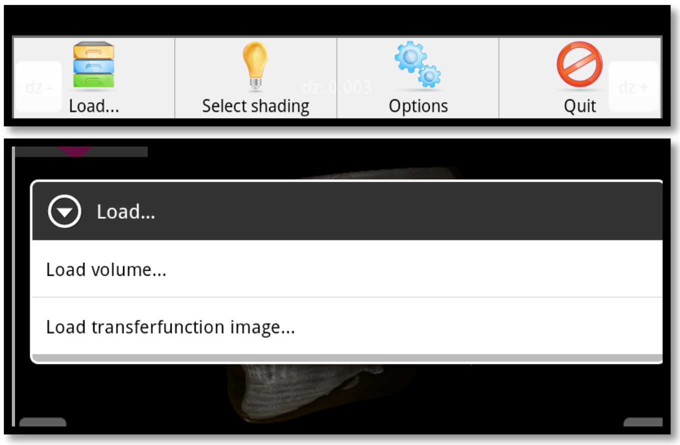

Figure 2: Menu and Loading screen

<u>MIP (Maximum intensity projection):</u> To calculate the volume the maximum intensity of the sight of view ray of the pixel is taken.

<u>Averaging:</u> The averaging amount of a ray will be taken by adding all the intensities up and divide them by maximum number of steps. It uses the transfer function for colour as well.

<u>Compositing:</u> Uses phong shading and a transfer function for the final image.

<u>Slicing:</u> One can slice through the volume in X Y and Z direction.

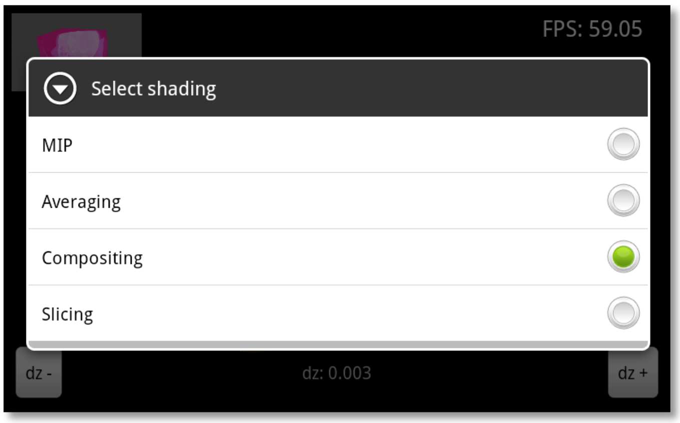

Figure 3: Shader menu

Next is the option menu. In the option menu FPS calculation and adaptive sampling can be turned on and off. The last item in the menu toolbar is the quit button. By pressing it, the program will be closed.

Touch input and adaptive sampling

The program supports two different gestures. One finger touch and movement to do a rotation in all 8 directions and two fingers for scaling. When the volume is touched, adaptive sampling will be activated. It tries to adapt the step size according to the current frame rate. This is done by simply dividing 60 (wanted frames) by the current frame rate times an appropriate sample distance of 0.005. This will only be done when interacting with the volume.

RESULTS:

The following volumes were tested with the program. The rendering was tested on a *HTC Desire HD* with an *Adreno 205* GPU with all 3 shaders, a step size of 0.002,adaptive sampling on and with a scale factor of 1

lobster.dat – A lobster
X:120, Y:120 , Z:34 - 979kb.

MIP:
Rotating (left/right) ~8.1 to ~9 FPS
Finalizing the image with the chosen step size (sight from the top): ~1.51 fps

Averaging:
Rotating (left/right): ~7.82 – ~8.1 FPS
Finalizing the image with the chosen step size (sight from the top): ~1.92 fps

Compositing:
Rotating (left/right): ~7.84– ~8.88 fps
Finalizing the image with the chosen step size (sight from the top): ~0. 78fps

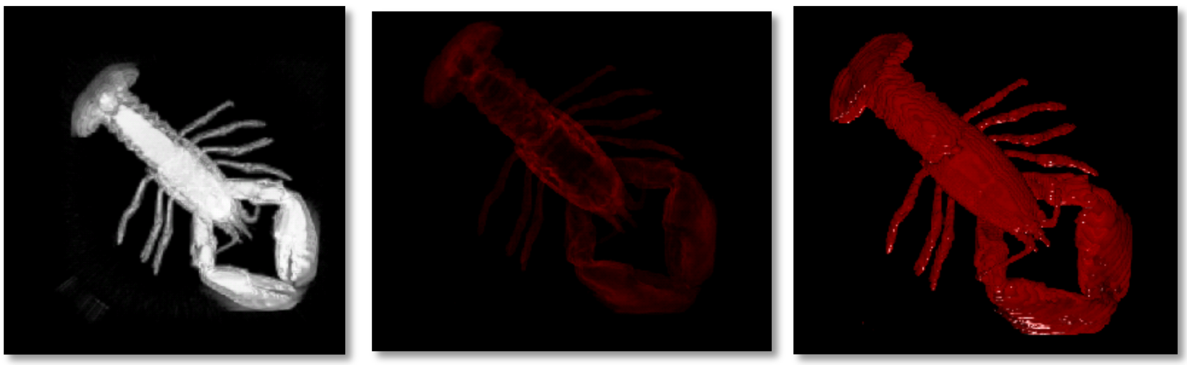
Figure 4: Lobster rendered using MIP, average and compositing

skewed_head.at – Human head
X:184, Y:256 , Z:170 – 9MB.

MIP:
Rotating (left/right) ~8.34 - ~8.94 FPS
Finalizing the image with the chosen step size (sight from the front): ~1.9 fps

Averaging:
Rotating (left/right): ~7.73 – ~8.1 FPS
Finalizing the image with the chosen step size (sight from the front): ~2.23 fps

Compositing:
Rotating (left/right): ~7.87– ~8.1 fps
Finalizing the image with the chosen step size (sight from the front): ~1. 35fps

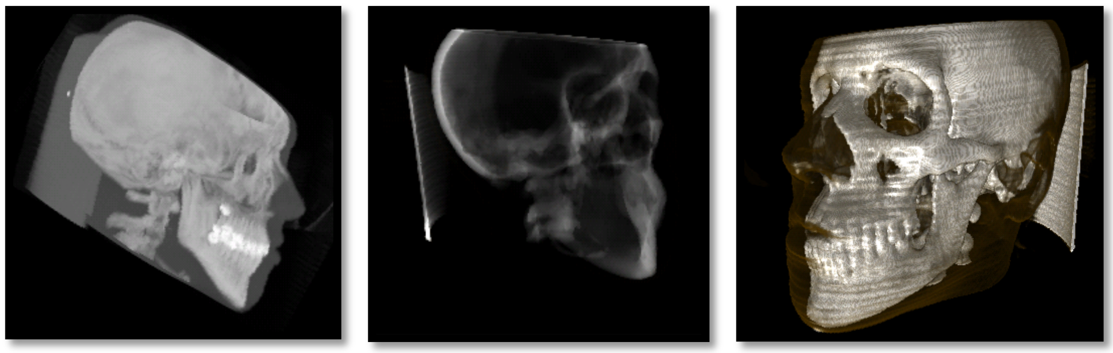
Figure 5: Skewed head rendered with MIP, average and composition. The last image was scaled as well

<u>stagbeetle.dat – A stag beetle</u>
X:277, Y:277 , Z:164 – 25MB.

<u>MIP:</u>
Rotating (left/right) ~7.7 to ~7.9FPS
Finalizing the image with the chosen step size (sight from the top): ~1.81 fps

<u>Averaging:</u>
Rotating (left/right): ~7.44 – ~7.76 FPS
Finalizing the image with the chosen step size (sight from the top): ~1.14 fps

<u>Compositing:</u>
Rotating (left/right): ~7.44– ~7.58 fps
Finalizing the image with the chosen step size (sight from the top): ~1. 22fps

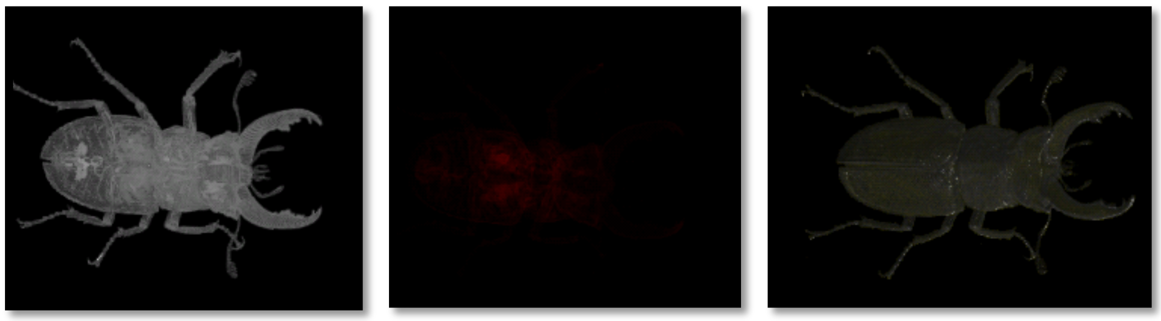
Figure 6: Stag beetle rendered with MIP, average and composition. Average image is very dark

PROBLEMS

One of the main issues was the OpenGL extensions of the mobile phone. First there were problems with the 3D textures methods. The linker was not able to find them. However simply defining the API entry manually was not enough. It is important to add the *-lEGL* lib to the *android.mk* file to do that.

Another problem was that non-power of two textures were not working even though the phone has the extension for non-power of two textures. After rescaling the volume texture it was possible to render the volume for the first time. However the texture slices where shifted (Figure 7). To avoid that it is important to use a power of two texture and unsigned int texture coordinates and not float coordinates otherwise one will encounter an interpolation problem of the texture coordinates and get shifted slices as a result.

The last problem concerns the shader. After observing a problem with a very high step size, the volume was not rendered properly and parts were missing (Figure 8), I found out that the iteration steps in the shader are limited to 256. To avoid this limitation a second "for" loop was implemented, which means performance loss.

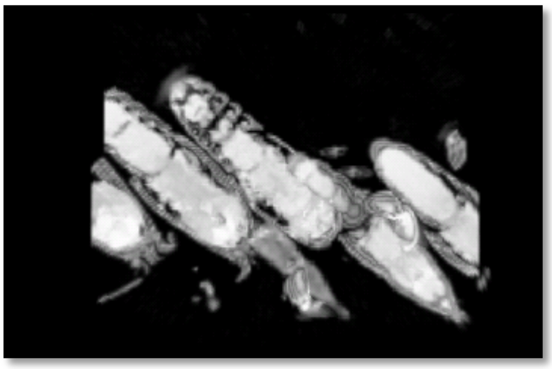
Figure 7: Shifted slices

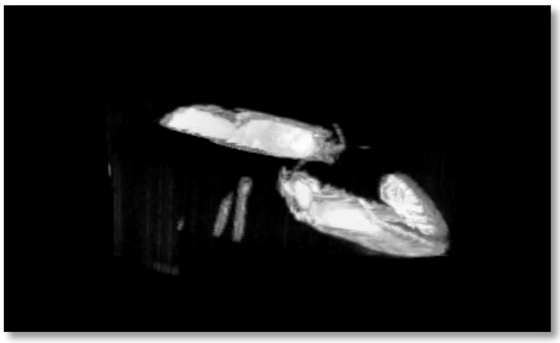
Figure 8: Iteration limitation

CONCLUSION

This project shows that it is possible to do a volume rendering in real time on a mobile device using different shaders and a transfer function with accurate speed. However it is still too slow for more powerful features and practical use. With faster hardware the program can be improved by adding an interactive transfer function menu. Another improvement could be to avoid the second loop in the shader or to completely avoid loops in it.

SOURCES

Files:
https://www.cg.tuwien.ac.at/courses/Visualisierung/Angaben/Bsp1.html

Display of Surfaces from Volume Data
Marc Levoy
IEEE Computer Graphics and Applications,
Vol. 8, No. 3, May, 1988, pp. 29-37.