School of Humanities and Informatics
Thesis Project in Informatics 30 credits
Avanced level
Spring semester 2011

Real-Time Audio Simulation with Implicit Surfaces using Sphere Tracing on the GPU

Peter Sjöberg

Real-Time Audio Simulation with Implicit Surfaces using Sphere Tracing on the GPU

Submitted by Peter Sjöberg to the University of Skövde as a dissertation towards the degree of M.Sc. by examination and dissertation in the School of Humanities and Informatics.

2011-06-21

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature: _____

Real-Time Audio Simulation with Implicit Surfaces using Sphere Tracing on the GPU

Peter Sjöberg

Abstract

Digital games are based on interactive virtual environments where graphics and audio are combined. In many of these games there is lot of effort put into graphics while leaving the audio part underdeveloped. Audio in games is important in order to immerse the player in the virtual environment. Where a high level of emulated reality is needed graphics and audio should be combined on a similar level of realism. To make this possible a sophisticated method for audio simulation is needed.

In the audio simulation field previous attempts at using ray tracing methods were successful. With methods based on ray tracing the sound waves are traced from the audio source to the listener in the virtual environment, where the environment is based on a scene consisting of implicit surfaces. A key part in the tracing computations is finding the intersection point between a sound wave and the surfaces in the scene. Sphere tracing is an alternative method for finding the intersection point and has been shown to be feasible for real-time usage on the graphics processing unit (GPU).

To be interactive a game environment runs in real-time, this fact puts a time constraint on the rendering of the graphics and audio. The time constraint is based on the time window to render one frame in the synchronized rendering of graphics and audio based on the frame rate of the graphics.

Consumer computer systems of today are in general equipped with a GPU, if an audio simulation can use the GPU in real-time this is a possible implementation target in a game system. The aim of this thesis is to investigate if audio simulation with the ray tracing method based on sphere tracing is possible to run in real-time on the GPU. An audio simulation system is implemented in order to examine the possibility for real-time usage based on computation time.

The results of this thesis show that audio simulation with implicit surfaces using sphere tracing is possible to use in real-time with the GPU in some form. The time consumption for an audio simulation system like this is small enough to enable it for real-time usage. Based on an interactive graphics frame rate the time consumption allows the graphics and audio computations to use the GPU in the same frame time.

Key words: Audio, Simulation, Implicit surfaces, Sphere tracing, GPU, Real-time

Contents

1 Introduction 3

1.1 Problem 4

1.2 Aim and Objectives 5

2 Background 6

2.1 Introduction 6

2.2 Audio Simulation Using Ray Tracing 7

2.3 Sphere Tracing 9

2.4 Audio Rendering on the GPU 10

2.5 Programmable Hardware and the GPU 11

2.5.1 Pipeline 11

2.5.2 Shaders 12

2.5.3 Input and Output 12

2.6 Audio Localization 13

3 Implementation 14

3.1 Overview 14

3.1.1 Setup 15

3.1.2 Sphere Tracing 15

3.1.3 Audio Output 16

3.2 Implementation Details 16

3.2.1 Platform 16

3.2.2 Setup 16

3.2.3 Sphere Tracing 17

3.2.4 Audio Output 20

4 Experiments and Results 21

4.1 Overview 21

4.2 Performance 23

4.2.1 Hardware 23

4.2.2 Scenes 23

4.2.3 Measurements 24

4.3 Audio Localization 26

4.3.1 Scenes 27

4.3.2 Test Subject Group 27

4.3.3 Question Form 27
4.3.4 Outcome 28
4.4 Analysis 29
5 Conclusion 31
6 Future Work 32
References 33

1 Introduction

Digital games are based on interactive virtual environments where graphics and audio are combined. In many of these games there is lot of effort and money put into graphics while leaving the audio part underdeveloped (Friberg & Gärdenfors, 2004). Audio in games is needed in order to immerse the player in the virtual environment as shown by Naef, Staadt and Gross (2002) and Funkhouser et al. (1998).

In digital games that need a presentation with a high level of reality emulation the presentation graphics and audio should be combined on a similar level. If the graphics part of the game uses methods to mimic real life phenomena the audio part should match this level of realism and use a simulation method based to some degree on physics.

The interactive nature of games demands that the game system runs in real-time, this aspect of the system puts a time constraint on the graphics and audio rendering. The time constraint is based around the frame rate of the graphics rendering where graphics and audio for each frame must be rendered in a fraction of a second. A real-time audio system is also usable in other areas where we have some degree of interactivity requirement.

Some audio simulation systems have previously used ray tracing methods based on the work of Whitted (1980) with some amount of success as in the work of Rindel (2000). Ray tracing methods give a straightforward way to compute audio propagation based to some degree on physics.

Ray tracing was primarily intended for graphics but as light waves and sound waves are both wave phenomena this method is suitable for audio as well (Funkhouser et al., 1998). The basic principle of the ray tracing methods is to calculate the sum of all incoming sound waves at a defined position in the scene. Virtual audio sources emit sound waves that affect a virtual listener position.

In the audio simulation a virtual scene is used which consists of geometry primitives arranged in a structured way. As described by Akenine-Möller, Haines and Hoffman (2008) polygon-based surfaces are commonly used in games for consumer hardware. Another way of describing geometry is through implicit surfaces that describe geometry through algebraic functions. The functions describe all the points on the surfaces and can be utilized for exact computations on continuous surfaces. Implicit surfaces also enable efficient intersection computations and efficient changes in topology (Akenine-Möller et al., 2008).

A method aimed to simplify ray tracing of implicit surfaces is the sphere tracing method proposed by Hart (1996). Distance functions to implicit surfaces are utilized in the sphere tracing method in an iterative process.

Consumer computers of today have dedicated hardware for graphics rendering commonly known as GPU (graphics processing unit). The graphics-rendering pipeline of the GPU is programmable with programs commonly known as shaders that enable general programming to some degree. Singh and Nayaranan (2009) used the GPU for ray tracing implicit surfaces and showed that “ray marching” methods like sphere tracing are suitable for the GPU in terms of performance.

Ray tracing based methods require time consuming computations (Akenine-Möller et al., 2008) that might not be possible to run in real-time on the CPU when a high level of detail is needed. If the GPU can assist the audio simulation by tracing sound waves

with the sphere tracing method the computation time of the simulation can be lowered. A lowered computation time makes the simulation usable in games and other interactive environments as it meets the time constraint of real-time usage. Shaders can be used in the GPU to handle parts of the audio simulation, the simulation system is then split between the CPU and the GPU and the strength of each hardware unit is utilized.

1.1 Problem

As argued above the audio in virtual environments is important and should match the graphics in the level of emulated reality. A method for audio simulation is needed to give a structured approach to render the audio to match the graphics. In consumer computer systems that are able to run real-time virtual environments like games it is possible to utilize the GPU for audio simulation and devote a part of the audio renderer to it. A method for audio simulation that suits the GPU well in terms of performance makes the rendering usable in real-time to some extent.

The audio simulation method used in this thesis is based on the ray tracing approach discussed by Rindel (2000) where sound waves are traced in the virtual scene from the audio sources to the listener. Ray tracing were originally proposed by Whitted (1980) as a method for graphics rendering but has been shown to be usable for audio as sound waves share some properties with light rays.

A basic step in the ray tracing approach is to find the intersection point between the sound waves and the geometry of the virtual scene. When the intersection point is found reflections and other phenomena can be simulated as the sound waves propagates in the scene and finally gathers at the listener position.

The sphere tracing method that is described in the background chapter gives an alternative method to find the intersection point for each ray. This method is suitable for the GPU in terms of performance and could be used to simulate audio in real-time. If sound waves can be traced through the game scene presented through the graphics there is a straightforward way of rendering the audio.

By using sphere tracing the audio simulation is limited to implicit surfaces that can be described by distance functions. To simulate audio in a typical game scene a separate scene is needed that is a simplified version of the original scene with less geometry. A typical triangle based game scene may contain millions of triangles that are not feasible to use with sphere tracing in real-time. The separate audio scene will act as a rough estimate of the real scene using implicit surfaces. Preparation wise this version must be created alongside the normal scene and will require efficient geometry using a small number of surfaces. Manual work may be needed to create a feasible audio scene from the original triangle geometry.

The distance functions that are used with sphere tracing can be used to clone surfaces computation wise. The distance function could for example with a simple modulo operation clone a surface in a certain axis. This gives an elegant way of “repeating” the same surface at a fixed distance without adding implicit surface data to the scene. The distance functions combined with the implicit surfaces also enables geometric transformations in an efficient way.

In a typical game that uses both graphics and audio the GPU could obviously only use a limited part of the total render time for a frame. The GPU cannot be devoted only to

audio unless the game is an audio only game. This fact needs to be considered when taking the real-time time constraint into consideration as the audio renderer only can use the GPU part of the time required to render a frame.

1.2 Aim and Objectives

With using sphere tracing assisted by the GPU this thesis aims to examine if it is possible to run an audio simulation in real-time. This is possible if an amount of rays can be traced enough to output a convincing audio result. Some variables in the system can be altered to balance between performance and audio quality. One of the variables is the predefined minimal distance that is used in the sphere tracing method and another is the amount of rays used. The nature of distance functions that makes it possible to “repeat” surfaces is usable on some type of scenes and can possibly improve the performance.

The contribution of this thesis is results that give a hint whether audio simulation is possible in real-time using sphere tracing on the GPU. This thesis uses mainly work from others and should be viewed upon as an investigation using a combined collection of methods, where the methods are derived from the work of others. Aspects of using ray tracing methods for audio simulation and definintion and implementation of the sphere tracing method are previous work by others that are here used in the same system and implemented on the GPU for real-time usage.

To fulfill the aim three objectives are completed in this thesis, the first objective is to carry out a literature analysis where background information in the audio simulation field is collected. The background information shows previous work in this field and relevant key concepts connected to the problem. Previous work also builds the base to the implementation in the next objective.

The second objective is to implement an audio simulation system in order to enable experiments connected to the problem. In this case experiments are able to generate measurements that help the investigation of the problem.

The third and last objective is to conduct experiments connected to the problem, in this thesis two different types of experiments are used. The main type of experiment uses the audio simulation implementation developed in the previous objective and the complementary type of experiment evaluates the implementation using human test subjects. The outcome of these experiments enables an analysis in which the aim can be fulfilled.

The following three chapters describe the output of the objectives mentioned above, these chapters are followed by a conclusion of the results and a discussion of possible future work.

2 Background

In this chapter the background information to this thesis is presented in order to motivate the work and introduce of key concepts used in the text. The key concepts are described in detail where needed and connected to related work in the audio simulation field.

2.1 Introduction

Digital games are often based on interactive virtual environments that use a combination of graphics and audio. A lot of effort is put into graphics but audio is still underdeveloped in many cases (Friberg & Gärdenfors, 2004). Naef et al. (2002) and Funkhouser et al. (1998) show that audio in games are needed in order to immerse the player in the virtual environment presented. Games can also be solely based on audio in audio only games where no graphics is used at all. Friberg & Gärdenfors (2004) claim that this type of game is suitable for people with visual impairments who cannot use the presented graphics. In this case the audio must be developed enough to make a worthwhile gaming experience.

In digital games where a presentation using a high level of emulated reality is needed, graphics and audio should be combined on a similar level. If the graphics part of the game uses methods to mimic real life phenomena the audio part should match this level of realism. A system or method for rendering audio that is in some way based on physics simulation is often needed to make the audio sound in a convincing and realistic way (Naef et al., 2002).

A system for audio simulation would logically be put in an audio renderer in the game system in the same way as Naef et al. (2002) used as part of their audio rendering pipeline. The audio renderer is focused on calculating the audio that affects the virtual listener in the game environment, where the virtual listener represents the position of the player. If the audio renderer can manage this in a convincing way the player would perceive the audio as “natural” which should make it easer to get immersed in the environment.

On the basic level the audio renderer must calculate all audio samples for each audio channel that should be affected. The number of samples needed per time slot is based on the sample rate and grows linearly with the number of channels. The renderer continually generates a stream of audio samples that is passed to the audio hardware.

To be interactive the game system must run in real-time, this means that several graphics frames must be rendered each second based on the current frame rate that the system wants to obtain. The frame rate must be high enough to make animations appear in a convincing way as the frame rate dictates how fluid the animations can be rendered.

The time constraint of the real-time game system affects both the graphics and audio rendering. To make the presented graphics match the audio the audio rendering should match the frame rate of the graphics. The time constraint on both the graphics and audio rendering is equal to the time needed to render one frame.

The time constraint on the audio renderer demands a simulation that is possible to run in real-time. A deep simulation of the audio is not possible as computations using a large number of audio phenomena with a high level of detail would be too time

consuming. An approximation or less complex methods can be used to enable real-time usage for the audio simulation, the simulation should be efficient enough to enable real-time usage but still be able to output a convincing result to match the graphics. A convincing audio output would emulate reality to the same level as the graphics.

The following sections in this chapter introduce the key concepts behind audio simulation in real-time using the GPU. The audio simulation in this thesis project is based on the ray tracing method that is described in the next section followed by a description of an alternative method for tracing rays. The latter part of this chapter motivates the GPU application and introduces audio rendering on the GPU. Lastly this chapter introduces concepts used in audio perception that is used in the audio experiments.

2.2 Audio Simulation Using Ray Tracing

To simulate audio in the renderer an audio simulation method is needed. Some audio simulation systems has previously used ray tracing methods with some amount of success (Rindel, 2000). Variants of ray tracing like beam tracing has also been shown to be feasible for audio simulation (Funkhouser, 1998). Ray tracing methods give a straightforward way to compute audio propagation based on physics.

In the audio simulation a virtual scene is used that consists of geometry primitives arranged in a structured way. It is often practical to arrange the geometry so that it mimics a small part of the real world like corridors in an office or a series of tunnels in a cave. The geometry can in theory be based on a mixture of basic primitives and general implicit surfaces.

Implicit surfaces give a way of describing surfaces using implicit algebraic functions. Some common basic surfaces include the quadrics that for example can describe a sphere, a torus and other shapes. Two advantages over polygon-based surfaces are efficient intersection calculations and efficient topology changes.

In a game the scene is based on the already established environment presented through the graphics. This way the audio simulation uses the same scene as the visual one and has the basic information to render a correct result. In a game that uses consumer hardware the geometry is based solely on triangles and the amount of triangles is usually great. The consumer graphics hardware can handle this amount of triangles in real-time but the audio simulation most probably needs a simplified version to meet the real-time demand. An example of this is shown in figure 1 where the original scene is too detailed and a simplified version is created here marked with dashed lines.

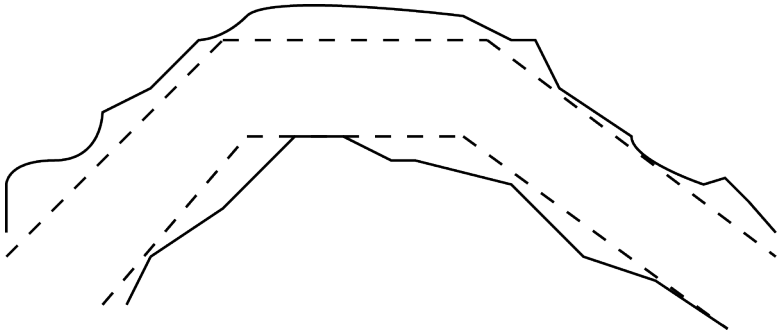

Figure 1. Example of a simplified scene resembling a cave tunnel. The continuous line segments represent the visual scene and the dashed line segments represent the simplified scene used in the audio simulation.

The basic principle of the ray tracing methods is to calculate the sum of all incoming sound waves at a defined position in the scene. The defined position is equal to the virtual listener that in games is equal to the position of the player avatar or the virtual camera. Ray tracing was primarily intended for graphics as first proposed by Whitted (1980) and is based on the fact that light rays are travelling through the air from light sources to the eye. As described by Funkhouse et al. (1998) light rays and sound waves are both wave phenomena, this fact makes global illumination methods for graphics usable for audio as well.

To use ray tracing for simulating audio virtual audio sources are defined in the scene. For each time slice in the simulation a number of sound waves are emitted from the audio sources that are traced through the scene as shown in figure 2. Sound waves that reach the virtual listener affect the audio and add up to make the final audio. Sound waves are traced through reflections and other physics phenomena to mimic real life sound propagation.

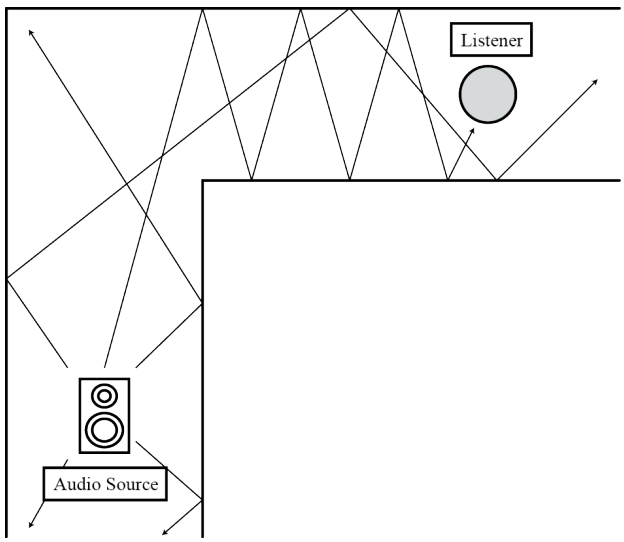

Figure 2. Sound waves are traced from the audio source to the listener. The speaker represents the virtual audio source and the circle represents the listener.

The basic problem with ray tracing is the time consuming calculations around the number of rays that has to be traced in the scene. The intersection between the rays

and the geometry must be found for each trace step where a trace step is a phenomenon that breaks the current direction of each traced ray. A maximum number of trace steps must be defined which directly translates to the number of iterations in the algorithm. The number of iterations is usually decided based on a balance between the quality of the result and the computation time.

If the scene geometry is based on implicit surfaces a variant of the ray tracing method can be used. Hart (1996) proposed the sphere tracing method that can be used to trace the rays in an alternative way.

2.3 Sphere Tracing

Sphere tracing is geometric method proposed by Hart (1996) aimed to simplify ray tracing of implicit surfaces and provides an alternative method to trace rays. In this method distance functions to implicit surfaces are utilized to "march" along each ray with varying distances. This process continues until the desired intersection point between the geometry and the ray has been found. A predefined minimal distance is used to stop the process and decide an intersection point. The process used in the sphere tracing method is shown in figure 3.

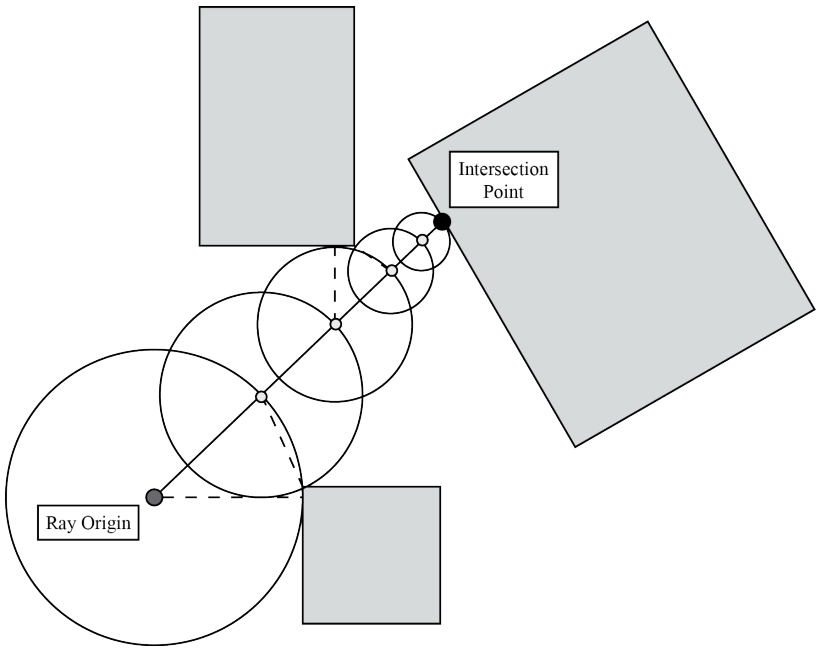

Figure 3. The sphere tracing method is "marching" along the direction of the ray. For each step the distance to the closest surface is found and is used as the "marching" distance. The gray boxes represent surfaces in the scene and the circles represent each "marching" step used in the process. The dashed lines represents the closest distance found in each step.

Based on the properties of the implicit surface used and the minimal distance the "marching" process may result in a non-exact result. This is an important distinction to ray tracing where an exact result is calculated for each trace step. Depending on what kind of results is desired the minimal distance can be adjusted to get more exact

results but with the cost of performance. The possibility to adjust the minimal distance to gain performance can be used in the real-time audio simulation.

2.4 Audio Rendering on the GPU

The consumer computer hardware including game consoles and mobile phones of today has dedicated hardware for graphics rendering. Sophisticated graphics hardware is commonly known as GPU (graphics processing unit) to mark the fact that these units of hardware have capabilities that are programmable. The consumer GPU is focused on rasterizing triangles and implements the latter part of the rendering process such as geometry transformations and complete rasterization.

As discussed by Knoll et al. (2007) and Singh and Nayaranan (2009) the GPU is well suited for algorithms like ray tracing where the computations are heavy but the communication between each part is less heavy. They also point to the fact that the GPU is more effective computation wise than the CPU if there is a little amount of branching and the memory access is not random.

The GPU is suitable to use for all computations that are in at least one form independent and can be parallelized as is the case with the graphics rendering pipeline. Singh and Nayaranan (2009) used the GPU for ray tracing implicit surfaces and showed that "ray marching" methods like sphere tracing are suitable to use with the GPU performance wise. Considering this previous work it is apparent that sphere tracing can be used on the GPU in real-time and in turn makes it usable for audio simulation. The GPU can assist the audio simulation in real-time by tracing rays with sphere tracing which makes in turn makes the concept usable in games and other interactive environments.

Games for consumers which want to present an immersive virtual environment with believable graphics and audio need to utilize the hardware that is available. If a GPU is available it should be used in order to get as much detail in the graphics as possible in relation to the time constraint put on by the real-time nature of the rendering. This is also true for the audio where an audio simulation may be implemented on the GPU to be able to render audio in real-time.

The GPU has some programmable capabilities in the graphics pipeline commonly known as shaders. Shaders allow small programs to substitute parts of the graphics-rendering pipeline to mainly give the possibility to change the graphics output. Shaders in today's consumer hardware are general enough to allow other types of calculations. The graphics pipeline is always in use but input and output to the program is possible through graphics buffers like rectangular images and screen pixels.

The shaders in the GPU can be used for audio simulation to handle parts of the system. The audio simulation system is then split between he CPU and the GPU and the strength of each hardware part can be utilized. Naturally the handling of data and dynamic calculations is be assigned to the CPU and sample rendering is assigned to the GPU.

2.5 Programmable Hardware and the GPU

The graphics hardware for consumers is created to accelerate graphics calculations and realizes a number of frequently used parts of the graphics rendering. Latter parts of a graphics rendering pipeline were supported but were fixed in their basic functions with limited configuration possibilites.

New demands on the graphics rendering forged the possibilty to make some parts of the pipeline programmable and the graphics hardware were coined GPU to mark this fact. As some parts of the pipeline are programmable to a certain degree it is also possible to use the GPU for calculations other than graphics. The programs in the GPU still use basic graphics structures for data input and output but the actual data is handled in a chosen way.

2.5.1 Pipeline

The graphics rendering pipeline used in the GPU is focused on rasterization of triangles and the related calculations. Programs in the GPU must therefore adhere to some restrictions of the actual pipeline being used. The parallelized nature of the pipeline also makes some algorithms and data operations impossible.

As presented by Akenine-Möller et al. (2008) the pipeline of the current consumer GPU can be divided into a number of steps as shown in figure 4. Not all steps are programmable based on the fact that some calculations are done in the same way no matter what is rendered.

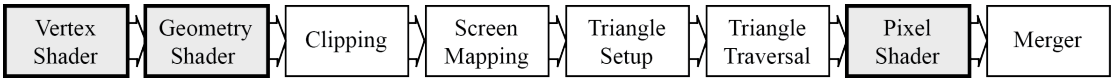

Figure 4. The graphics pipeline of the GPU is divided into a number of steps. Programmable steps are marked with a shade of gray.

The fully programmable pipeline steps are commonly known as shaders and are marked in figure 4. These steps have different tasks and are viwed upon as three different ways of programming the GPU. The steps also hold different limitations as to what operations are possible due to their placement in the pipeline.

An important aspect of GPU programming is to balance the use of the three pipeline shader steps as they are used in different amounts for a given rendered scene. The vertex shader typically runs once for each triangle vertex while the pixel shader runs for each pixel in each triangle. Geometry shading is done for each triangle and is by comparison the least executed type of program. This shader is also optional to program and can be left out if no geometry manipulation is needed.

Another important aspect of GPU programmering is that input and output between the pipeline steps is limited to the basic parameters defined in the pipeline. The parameters are connected to graphics rendering and are limited in number and data types. The GPU has a predefined set of named parameters that can be used in each shader and cannot be extended.

2.5.2 Shaders

The shader term commonly describes a program used in one of the programmable pipeline steps and is further described by Akenine-Möller et al. (2008). A shader is viewed upon as package that contains the program code and descriptions of expected input and output connected to other pipeline steps. Some shaders also depend on external data that is accessed in the code and therefore can be viewed as a part of the shader package.

Shaders use special hardware instructions that can perform common math operations commonly used in graphics. An instruction can for example multiply two vectors in one atomic operation. High-level languages for shader programming are available that hides som of the details like registers and actual instructions. The constructions used in the high-level languages often have a direct translation to the hardware instructions, this fact makes these languages feasible performance wise.

Some alternatives for programming the GPU exist like *CUDA* and *DirectCompute*. These API:s enable general programming on the GPU through regular programming languages with some restrictions. At the time of writing this thesis these alternatives are either bound to a specific hardware manufacturer or a specific operating system. Using shaders is the established method for GPU programming that has either direct or indirect translations on different hardware and operating systems. This method of programmering the GPU is explained by Akenine-Möller et al. (2008) and has been used in previous work that involves the GPU like Knoll et al. (2007) and Singh and Nayaranan (2009).

2.5.3 Input and Output

As shaders are bound to the graphics rendering pipeline some restrictions apply for input and output data. Aside from a limited amount of constant variables the method for accessing data for graphics rendering consists of image data either as input textures or output screen pixels. A texture is an established term used in graphics rendering that describes the memory data of a rectangular image. Each element of a texture is called texel that is normally defined by a number of color components.

Shaders have the possibility to sample textures in a number of ways and output rendered pixels to a given destination. The destination for the rendered pixels is usually the visible screen but can also be set to an off-screen texture where the pixels are stored as texels.

By using textures it is possible to pass general data to a shader and by changing the destination for the rendered pixels the output can be used for later processing. As textures are still created and handled for graphics usage the data must be “packed” into the texel data of the textures. Textures in the GPU can use a number of different data types to store each texel, this gives the possibility to use a data type that will preserve the precision of the values between the CPU and GPU when general data is passed in textures.

Textures that are used in shaders can be viewed upon as a memory buffer for general storage but an important aspect of this is based on how the shaders sample the texture and output the data. The sampling of a texture always returns a texel that in turns give four values, the output of the shader is always a single pixel that is stored as a texel in

a destination texture. To read an arbitrary number of values and write an arbitrary number of values in a shader is often not possible.

2.6 Audio Localization

The ability to position an audio source for a human is called localization that is a term described by Blauert (1997). A number of factors surrounding the human ears and the way the sound is processed is used to position audio sources in the real world. The way that humans position audio sources affect the audio simulation system because the output of the renderer must in some ways adhere to these theories.

Neuhoff (2007) concluded that some frequency ranges are more suitable than other for the human ears to localize an audio source. Earlier experiments have shown that the frequency range from 200 to 1000 Hz is most suitable for audio localization. The experiment on audio source positioning in this thesis project should mainly use sounds in this frequency range as to make the audio sources used in the experiment optimal for localization.

3 Implementation

In this chapter the implementation of the audio rendering system is described that was introduced in the introduction chapter. As argued in the introduction chapter this system enables an evaluation of an audio simulation based on sphere tracing. This implementation is unique for this thesis but is based on others work. Others previously established the principle of an audio rendering system but this implementation introduces the sphere tracing method as the base for sound wave tracing with the GPU.

This implementation is based on others work including using the ray tracing approach for audio simulation (Rindel, 2000) and the sphere tracing method (Hart, 1996). Others work also contributes to the base for the GPU implementation with the rendering pipeline of Raghuvanshi et al. (2007) and the implementation details of Knoll et al. (2007) and Singh and Nayaranan (2009).

The rendering system is first described at a higher level by showing how its relevant parts are connected. The high-level description is followed by implementation specific descriptions of these parts to give a more complete picture of the system.

3.1 Overview

The rendering system naturally involves both the CPU and the GPU in the audio rendering but in time consuming terms the main processing is done in the GPU. As in the work of Singh and Nayaranan (2009) the main role of the CPU is to setup and prepare data for the sphere tracing based on the dynamic information related to the current scene. The CPU also handles indirect communication between the GPU and the audio hardware, as the CPU is the hub for hardware communication the GPU is not able to communicate with the audio hardware directly.

Each rendering frame of the system is divided into three logical parts that handles different tasks in the system and also gives logical boundaries between the CPU and GPU processing. The three parts are shown and named in figure 5.

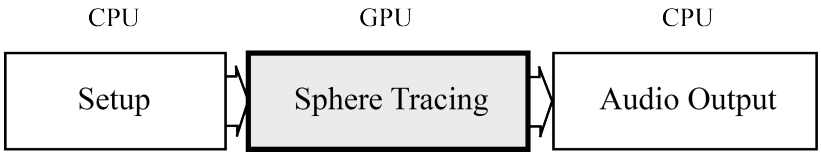

Figure 5. Each rendering frame of the audio rendering system is divided into three logical parts that handle different tasks.

In time consuming terms the second part is most relevant as all the sound wave tracing are computed here. This part is also done entirely on the GPU with the least possible amount of interaction with the CPU.

3.1.1 Setup

The first part of the rendering system handles the dynamic information in the system needed for the sound wave tracing. The dynamic information consists of data that describes the current scene environment. Relevant data for the environment is in this case the scene geometry, view transformation and positions for the camera and audio sources. This data is used to initialize the sound wave tracing to match the current scene that is rendered, this way both the audio and graphics rendering will use the same configurarion. Some important constants used in the tracing are also initalized like the minimum and maximum tracing distance used in the sphere tracing.

This part of the system also controls the GPU throughout the whole rendering with a number of passes that includes initializing and starting the corresponding shader code in the GPU. In addition a number of technical details are handled to make the GPU pipeline usable for audio rendering.

3.1.2 Sphere Tracing

The second part in the rendering system handles the actual sound wave tracing using the sphere tracing method. This is realised with a number of rendering passes based on the number of tracing iterations set for the wave tracing, the tracing computations are here done with the least possible interaction with the GPU as the time should be used efficiently without delays in the computations. This method of rendering with passes uses a setup where the output of each pass is fed as input into the next pass as shown in figure 6. This solution gives a natural way for each sound wave in the tracing to use the previous intersection point as origin and is similar to the rendering system used by Purcell et al. (2002).

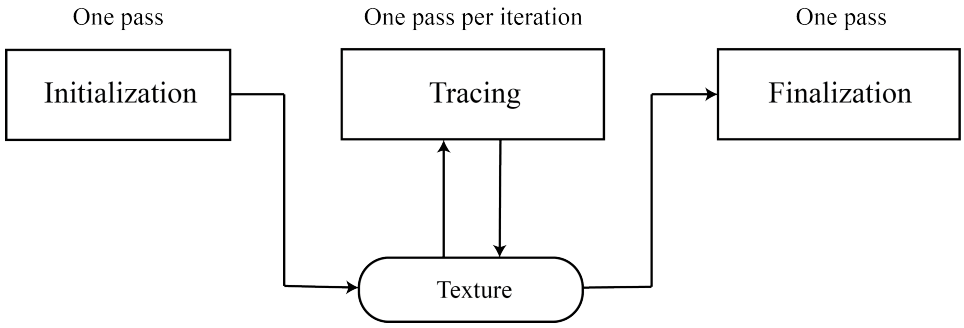

Figure 6. The sound wave tracing is done in one pass per iteration where the input data changes place with the output data for each pass.

Two more passes are also used in this part to initialize and finalize the sound wave tracing. The initialization pass computes the origin and direction for each sound wave based on the position of the audio source and a randomized direction. The pass for finalization prepares the audio output by computing how much the sound waves are contributing to the listener position.

The output of the sphere tracing part is the final intersection point for each sound wave traced in the scene, the final intersection point can be found in any of the iteration passes. For this system the most interesting information are the intersections

with the listener position as this contributes to the audio stream and is passed on in the system for further processing.

3.1.3 Audio Output

The third and final part of the rendering system renders the final audio stream based on the output of the sphere tracing part. Each rendering frame generates a small buffer of audio samples that is immediately passed on to the audio hardware for playback in real-time.

The contribution to the listener position from the sound waves are used to set relative values for the left and right channel of the audio output. The finalization is done in the CPU as a dynamic algorithm is used to gather the listener information that changes each rendering frame based on the number of sound waves reaching the listener. This algorithm is described in further detail below.

Similar to the rendering system used by Naef et al. (2002) the finalized output of this part consists of two stream buffers of audio samples, here the two streams represent the left and right audio channel. Each buffer is adapted for the audio hardware for immediate playback.

3.2 Implementation Details

This section describes some implementation details in the rendering system to give a low-level picture of the system. These implementation details indirectly show some choices made in the interaction between the CPU and GPU that is interesting in the domain of real-time rendering. The technical platform is described followed by a description of the relevant details in the three parts of the rendering system.

3.2.1 Platform

For this implementation the *DirectX* API is used to access the GPU and implement the shaders. The shader code is written in the high-level shading language that is part of this API. Code snippets that are shown below are using this language, this should be kept in mind when examining the shader code. As previously discussed the shader implementation has a direct on indirect translation to other API's for GPU usage.

In terms of the *DirectX* shader capability versioning system the version 3.0 profile was used when compiling and using the shaders in this implementation. Of the shader types available in the pipeline only the pixel shader is used for the actual sound wave tracing.

3.2.2 Setup

The first part of the rendering system that initializes the tracing has the task of passing the relevant information from the interactive scene to the GPU. The information is transferred to the GPU using global variables that are accessible in the shader code, these variables are in technical terms using the constant registers available in the GPU

to set global information that is uniform across all vertices and pixels in a rendering pass.

3.2.3 Sphere Tracing

This part of the implementation uses pixel shaders to be able to access each individual texel and interpret it as four data values. In GPU rendering terms this is accomplished by rendering two triangles that occupies the whole screen space where one texel in the texture match one execution of the pixel shader. The vertex and geometry shader types are not used here as the triangles are stated using pre-transformed screen coordinates.

As the pixel shader is only capable of outputting one texel per texture the only way to enable the output of more data is to use multiple textures. As the sphere tracing implementation needs to output more than four values each pass it utilizes two textures. The textures hold the minimum amount of data to represent each sound wave currently traced, the textures are sampled with the same texture coordinate for each texel and so the two textures are always aligned and read as a pair. This approach of data storage is used by the ray tracing system implemented on the GPU in the work of Purcell et al. (2002).

For each wave the first texture holds the origin position and a status of the tracing while the second texture holds the direction vector. This information is stored by "packing" values in the texels of the textures as shown in figure 7.

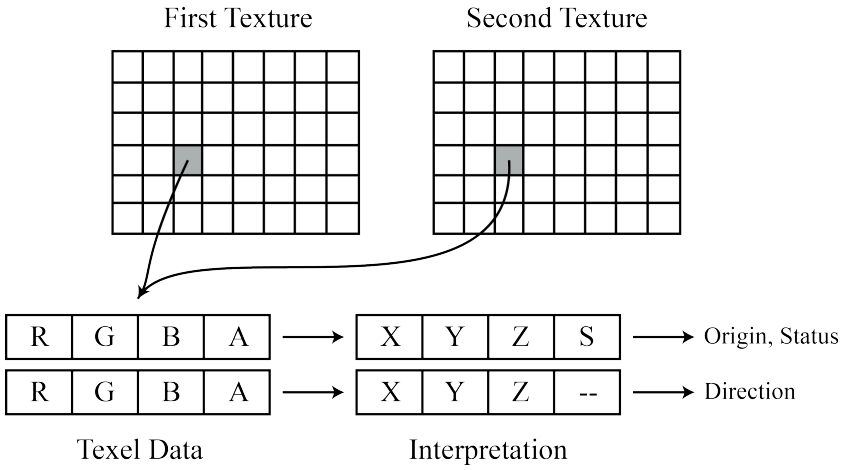

Figure 7. The tracing implementation uses two textures for storing the sound wave information and "packs" the values in the texels.

The textures are using 32-bit IEEE floating point format to enable an amount of accuracy in the computations, the precision of the values when transferring the data between the CPU and GPU is also preserved. To avoid texture-sampling artifacts the sampling in the shader is set to nearest sample without filtering.

As previously shown in figure 6 the first step of the tracing part is to initialize the data needed. The sound waves are here created and are based on the origin of the audio source and a random direction. A randomized direction in all three axes is used to

spread the sound waves in all directions around the audio source. The output of this step is two textures that hold the new origin and direction for each sound wave.

The main computation is handled by the sphere tracing step in which all iterations of the tracing are realized through one pass for each iteration. The computation for the tracing step is a direct translation of the algorithm proposed by Hart (1996). In each pass the new sound wave origin and direction is read from the input textures and when each pass is complete a new origin and direction are stored in the output textures. These new values are computed from the intersection point of the wave, the new origin is equal to the intersection point and the new direction is equal to the reflection vector at the intersection point based on the surface normal. This gives a straightforward way to use a "ping-pong" method for the input and output textures where they simply switch place for each pass.

A well-known problem with ray tracing methods lies the actual tracing computations in respect to phenomena like reflection and reftraction. As the new origin is equal to the intersection point the next iteration of the tracing is likely to find the same intersection point again (Glassner, 1989). A method aimed to remedy this problem is suggested by Glassner (1989) and implemented in this system, where the wave origin is offseted by a small amount before the tracing begins. The offset is based on the direction of the wave and uses a small constant.

Aside the origin position a status value is also stored, this value is used to mark the status of the tracing computation. This is an important value used in the iteration passes to avoid unnecessary computations as discussed by Purcell et al. (2002). A special value in the status marks that no surface in the scene has intersected with the sound wave and another special value marks that the listener sphere surface at the listener position has been intersected. These are two cases that need to be handled in the system, as the tracing should not continue. When no intersection is found the wave is considered "dead" for the audio simulation purpose. When an intersection between the wave and the listener surface has been found the tracing is also stopped for that particular wave, as this is the main purpose of the tracing computations. A status value that marks the listener surface is also used in the last step where the audio information is finalized for further processing.

The entry function code for the sphere tracing is shown in figure 8. This is the pixel shader code that is executed in each iteration pass and for each sound wave. The status value is used to skip further tracing and is also set by this function if no intersection with a surface as found.

In this function the sound wave data is first read from the input textures and the first selection decides whether the wave has intersected with the listener surface or not. If this is the case no tracing is performed, if not the tracing is performed for this sound wave. The tracing uses the origin and direction of the wave and offsets the origin as described above, in this case an offset of 4 % based on the normalized direction is used. This constant assures that the wave does not intersect with the same surface again as the distance function will not find the same surface in this iteration.

When the tracing is complete the function continues with a computation to reflect the sound wave at the intersection point, this is only done if the wave intersected with a surface. A selection is able to distinguish between these two cases by examining the result from the tracing. To make it possible to compute the direction of the reflected wave the surface normal must be known, the normal is computed by a function that averages the distances of a number of waves close to the intersection point. This

method of computing the normal is used to make the reflection independent of the surface and was proposed by Hart (1996) and used by Singh and Nayaranan (2009) in their GPU implementation. The independence from the surface is usable when implicit surfaces with different distance functions are used in the scene.

Lastly the tracing function outputs the new origin and direction for the sound wave, these values are based on the selections described above. These values will also be fed into the same function in the next iteration pass as described in the implementation overview.

```
void trace
(
	float2 inScreenPos : TEXCOORD0,
	out float4 outOrigin : COLOR0,
	out float4 outDir : COLOR1
)
{
	float4 waveOrigin = tex2D(rtTexture0Sampler, inScreenPos);
	float4 waveDir = tex2D(rtTexture1Sampler, inScreenPos);

	if((waveOrigin.w > -0.1) && (waveOrigin.w < 0.1))
	{
		outOrigin = waveOrigin;
		outDir = waveDir;
	}
	else
	{
		float object;
		float3 eps = waveDir.xyz * 0.04;
		float t = traceWave(waveOrigin.xyz + eps, waveDir.xyz, object);
		float3 ip = waveOrigin.xyz + t * waveDir.xyz;

		outOrigin = float4(0,0,0,-1);
		outDir = float4(0,0,0,0);
		if(t >= 0)
		{
			outOrigin = float4(ip, object);
			float3 normal = traceNormal(ip);
			outDir = float4(reflect(waveDir.xyz, normal), 0);
		}
	}
}
```

Figure 8. The entry shader function code for the sphere tracing step.

Sphere tracing utilizes a distance function to “march” along the direction of the sound wave, in this implementation a basic function is used that finds the closest intersection to all surfaces in the scene. The implementation code for the “marching” is shown in figure 9 and is a direct implementation of the algorithm proposed by Hart (1996).

```
float traceWave(float3 waveOrigin, float3 waveDir, out float object)
{
	float t = 0;
	while(t < maxDistance)
	{
		float distance = distanceFunction(waveOrigin + t * waveDir, object);

		if(distance < minDistance)
			return t;

		t += distance;
	}
	return -1;
}
```

Figure 9. The shader function code to “march” along the direction of the sound wave.

The last step of the sphere tracing finalizes the audio computations by iterating through the origin data for each wave and only take waves into account that has intersected with the listener. To know which waves have intersected with the listener the status value is used. Combined with the listener direction a contribution factor for the left and right channels for each wave is computed and stored in an output texture. The main function of this step is to prepare the data for further processing by the CPU by handling all possible computations that can be handled by the GPU given the available data.

3.2.4 Audio Output

The final part in the rendering system connects the sphere tracing to the audio hardware by using the output from the last step of the sphere tracing. The output of the sphere tracing audio preparation step is a texture that is here copied from GPU memory to CPU memory. This is the only time where data is copied from the GPU to the CPU in the system, this is important to note as the performance cost for this data transfer is noticeable and should be avoided when possible.

After the data is copied the contribution for all sound waves that intersected with the listener surface is summed and a final panning value is computed. The sphere tracing part as previously described marks the contributing sound waves in the texture through the status value. In addition to the sphere tracing renderer the sound system is constantly delivering audio samples to the audio hardware based on a monophonic audio source. The final panning value is used to mix the audio source so get proper levels on the left and right channel, the mixing uses an equal-level method between the channels.

4 Experiments and Results

In this chapter the experiments conducted in this thesis are described in an overview section, the details of the experiments are then presented and discussed in separate sections. Lastly the results of the experiments are summarized in an analysis section that compares the results to the purpose of the audio simulation in the real-time domain.

4.1 Overview

This thesis examines the possibility to use sphere tracing for audio simulation in real-time using the GPU. To do this an implementation for a basic sphere tracing system must be written which can run a basic simulation in a given scene. The audio renderer must also be incorporated into this system and be able to output a correct audio stream. The system must be able to automatically run sequence of tests with variable input and output relevant data. Relevant data in this case is the audio stream and aspects of CPU and GPU performance as the focus of this thesis project is real-time usage.

The performance of the CPU and GPU is measured as time consumed for the computations of one frame, where less time means higher performance. When the computations take less time a more detailed simulation can be used that utilizes a higher number of sound waves. Less computation time also makes it possible to implement more audio related phenoma in the simulation but still make it feasible for real-time usage.

The CPU and GPU performance can be compared against variables in the audio simulation as previously suggested. Apparent variables are the amount of sound waves that are traced in the scene and the “marching” minimal distance used in the sphere tracing method. With this data it is possible to draw conclusions about how feasible this method is for real-time usage.

A number of different scenes must be used in the measurements to give a better hint of where the method is feasible and show that the method is usable in different types of environments. An open scene compared to a scene with obstacles between the audio source and the listener is two examples of contrasting types of environments.

This investigation stemmed from the problem description aims to give a hint about the possibility of real-time usage, a simple approach is therefore used in the system. A complete implementation that can simulate audio in a scene and output audio is needed but it is only required to use basic sound wave tracing. Sound related phenomena like energy and head-related transfer are not part of this implementation as this can be implemented at a later stage and does not contribute to the aim of this investigation.

It is important to also examine the audio stream because one main motivation of this thesis is to give a more developed audio output. A systematic method for examining the audio is to replicate a real environment in a virtual scene and use real recordings from the real environment and compare them to the audio output of the renderer. This method needs a robust way of comparing the audio information. A sample-by-sample comparison approach will most probably not work in this case because the difference in recording methods. The real recording will be affected by the microphone

characteristics and an amount of noise that makes the recording differ in clarity compared to the output of the renderer. A more sophisticated method is needed where audio analysis is used to compare the information, this is however out of scope for this work.

A more realistic way of examining the audio output is to conduct a complementary experiment where the audio output is verified with humans. The primary function of this experiment is to find out whether the audio output is functional enough for humans to position the audio source in the scene. The audio simulation system is used in the experiment to present the current scene with graphics and the test subjects will try to position the audio source related to the presented scene. For the experiment it is possible to use different variable settings and find out if and how these settings change the perceived audio and the positioning of the audio source.

The main variables that affect the audio simulation to a high degree are the number of sound waves traced from the audio source and the number of iterations in the sound wave tracing. These variables indirectly decide the maximum distance the sound waves will be traced and the amount of sound waves that will reach the listener. In this case a larger distance and a larger amount of sound waves is assumed to enable more precise computations.

The outcome of the experiement gives an indication of the validity of the audio output in the audio simulation, this is important as the audio simulation should work in real world usage involving humans. If the indication shows that the audio simulation does not work in a real world usage the performance tests of the system are less relevant as the one main point of this thesis is to find a method that provides a higher level of realistic audio emulation.

The basic structure for this experiment is to handle one test subject at a time in a quiet environment and conduct a psychometric experiment which is a type of procedure discussed by Blauert (1997). Each person is introduced to a computer that has the test system running which is rendering a visual scene. In this scene a virtual audio source is used and the audio simulation is running in real-time and is rendering an audio stream. The test subject is instructed to sit in front of the computer and look straight into the screen it in order to align the ears with the virtual listener in the scene. The virtual listener direction is equal to the camera direction used to render the visual scene.

The test subject uses headphones to get a clear audio path from the system to the ears, this should be compared to using speakers which is more sensitive to placement and room acoustics. In this experiment a question form is used that focus entirely on the position of the audio source in two different types of scenes. If the test subjects are able to position the audio source in the scenes the audio stream from the renderer is valid. Even if the system has the possibility to move the audio source and the listener in real-time the experiment uses only static positions in order to only focus on the basic positioning. Only one audio source is used in the experiment to refine the result with avoiding interference between multiple sources and avoid conflicting audio information.

In the experiment two different sounds are used in the audio source to minimize the effect of the nature of the sound itself. The sounds use mainly the frequency range optimal for audio positioning explained in the background chapter but differ in context. An example of two different sound contexts is speech and a sinewave where

the speech could carry information besides the actual sound and the sinewave is a monotone sound without information.

The test subject group in this experiment should be have a degree of experience with digital games and virtual environments, as the setup is basic and requires some experience with using virtual environments. In this case the experience allows the test subjects to grasp the concept of a virtual scene rendered in three dimensions and localize audio using only headphones.

4.2 Performance

The performance of the audio simulation system is measured as time consumed for the computations of one frame, where less time means higher performance. Both the CPU and GPU are part of the simulation but the main computation time is spent in the GPU. As discussed in the overview section computations that take less time enables a more detailed audio simulation that is still feasible for real-time usage.

This section presents the performance measurements in different setups and with different variables used in the audio simulation system. First the hardware used to run the tests is described followed by description of the types of scenes used in the tests. Lastly the measurements of the tests are presented and discussed through a number of table figures.

4.2.1 Hardware

Two hardware setups were used in the performance tests to compare GPU types of different manufacturers in order to reduce the risk of local measurement errors stemmed from local details like driver configuration or lack of hardware implementation features. The two setups also differ in application and power usage as the first one is created for mobile usage with low power consumption and the second one is created for desktop usage with no obvious limit on the power consumption. Intended usage and power consumption affects both the CPU and GPU as the hardware construction of these units differ in expected performance.

The mobile hardware setup used a *Intel Core i3* CPU with two cores running at the clock speed of 2,13 GHz and a *Nvidia GeForce GT 330M* GPU running at the clock speed of 475 MHz.

The desktop hardware setup used a *AMD Athlon 64 X2* CPU with two cores running at the clock speed of 2,5 GHz and a *ATI Radeon HD3650* GPU running at the clock speed of 725 MHz.

4.2.2 Scenes

As described above two different types of scenes were used in the tests that differ in complexity due to different collections of surfaces. The two scenes that were used are shown in figure 10 and are named for further reference in this text.

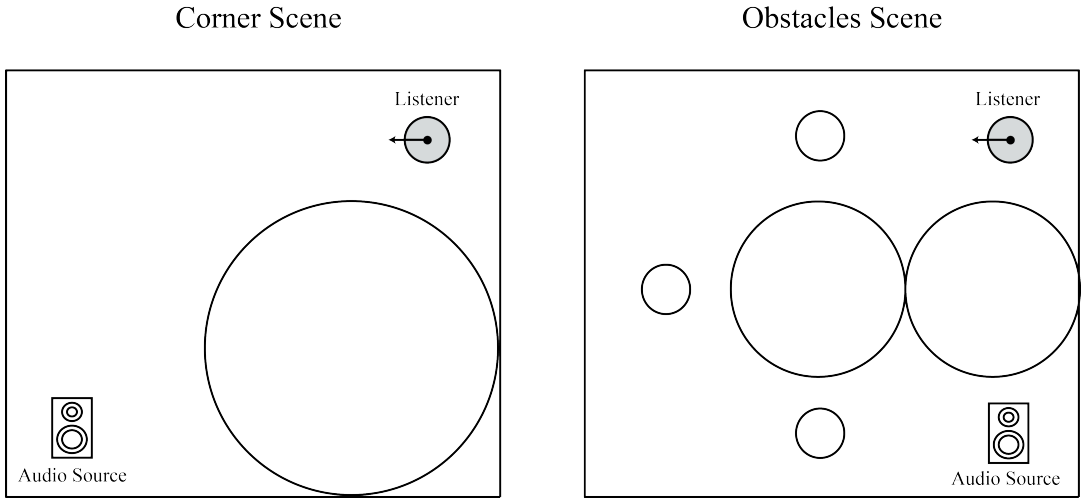

Figure 10. Two different types of scenes were used in the performance tests that differ in complexity, the scenes are shown as slices in the X-Z plane.

The corner scene only holds surfaces enough to build a basic "wall" structure and the obstacles scene adds more surfaces to place virtual objects between the audio source and the listener consisting of more of the "wall" structure and cylinders acting like "pillars". The scenes are defined in three dimensions but are shown here only as slices in the X-Z plane that represenents a view from above.

4.2.3 Measurements

The performance tests of the audio simulation system was conducted with the main variables of the tracing set to fixed values while the number of sound waves were increased. For each step of increasing wave count a timing value and the amount of intersections with the listener were noted. The timing value was noted as milliseconds per frame and the intersection count was noted based on a counted integer. Each set of tests is shown in figures 11.1 to 11.4.

The time consumed for a frame in a frame rate of 30 frames per second is roughly 33.3 milliseconds, this value should act as a reference when evaluating the measurements of the system.

For each test the minimum tracing distance was set to 0.1 world units and the maximum tracing distance was set to 300 world units. These values were set relative to the scene data that were defined with the extents of 30 world units in the X and Z direction and 20 units in the Y direction. The minimum distance was set to enable the tracing to find surfaces in the scenes to a degree of precision but still avoid unnecessary computations in each iteration pass where the "marching" steps would increase in number. The maximum tracing distance was set to enable the sound waves to travel the maximum distance between the scene extents in one iteration.

An analysis of all performance measurements can be found as part of the analysis section below. The performance test in figure 11.1 uses the corner scene and shows that a small amount of the traced sound waves reach the listener surface as expected. The time consumption for each frame allows for a frame rate higher than 30 frames per second when around 100000 sound waves are used. The mobile and desktop hardware setups show a similar time consumption curve with an increasing number of

sound waves, this shows that the measurements are not bound to the local hardware configuration or implementation details in the GPU. Both hardware setups are able run the audio simulation in real-time with a moderate amount of sound waves. When 1048576 waves are traced the GPU in the mobile setup did not handle the simulation system properly which can be seen in the odd time consumption value, this value can be ignored for the purpose of the test. The high resolution of the textures is the assumed technical reason for this behaviour.

Sound Waves	Time, Mobile	Time, Desktop	Listener Intersections
64	1.3	1.9	0
256	1.4	1.7	2
1024	1.5	2	13
4096	2.2	2.8	27
16384	4.1	6.6	95
65536	12	19.5	428
262144	40.5	73.2	1642
1048576	22	276	6760

Figure 11.1: Performance test table for the corner scene with a maximum of 4 iterations in the sphere tracing.

Compared to figure 11.1 the measurements in figure 11.2 show the impact of the increased number of iterations used in the sound wave tracing. A higher number of waves reach the listener surface that is expected when the number of iterations is doubled. Here the number of iterations also affects the time consumption as more sound waves are traced that has no effect on the audio output, as a result less sound waves can be used for real-time usage. The hardware setups show a similar pattern to figure 11.1 including the odd value in the mobile setup for 1048576 sound waves.

Sound Waves	Time, Mobile	Time, Desktop	Listener Intersections
64	1.4	2.4	0
256	1.6	1.9	6
1024	1.8	2.3	20
4096	2.8	5.5	63
16384	5.65	10.1	199
65536	19	33.9	848
262144	64.5	128.4	3270
1048576	20.5	500	13402

Figure 11.2: Performance test table for the corner scene with a maximum of 8 iterations in the sphere tracing.

In figure 11.3 and 11.4 the obstacles scene is used with the same difference in the number of iterations as the measurements of the corner scene. The same indication of a large number of sound waves not intersecting with the listener surface is shown here. As this scene contains more surfaces and a number of surfaces are blocking the path between the audio source and the listener the time consumption for the audio simulation is higher in this scene. A higher number of iterations are also needed for

the sound wave tracing to reach the listener surface, this is apparent in figure 11.3 where no intersections were noted. As with figure 11.1 and 11.2 the same pattern of time consumption between the hardware setups is apparent including the odd value when using 1048576 sound waves.

Sound Waves	Time, Mobile	Time, Desktop	Listener Intersections
64	1.39	1.7	0
256	1.51	1.8	0
1024	1.6	2.2	0
4096	2.42	3.5	0
16384	5.01	8.7	0
65536	16.55	30	0
262144	52.29	110	0
1048576	22	428.01	0

Figure 11.3: Performance test table for the obstacles scene with a maximum of 4 iterations in the sphere tracing.

Figure 11.4 shows that a small amount of sound waves reach the listener surface when a higher number of iterations is used, in this case the number of iterations is doubled compared to figure 11.3. As with the corner scene the time consumption is also increased as a direct result of the increased number of iterations. Similar to the previous figures 11.1-11.3 the hardware setups show the same pattern of time consumption.

Sound Waves	Time, Mobile	Time, Desktop	Listener Intersections
64	1.5	1.8	0
256	1.81	2	0
1024	2.1	2.6	0
4096	3.26	5	0
16384	7.8	13.8	1
65536	24.4	51.3	5
262144	86.1	194.6	25
1048576	22.4	764.3	84

Figure 11.4: Performance test table for the obstacles scene with a maximum of 8 iterations in the sphere tracing.

4.3 Audio Localization

This section describes the implementation of the complementary experiment aimed to give an indication of the validity of the audio output as described above. To give an overview of the experiement the details about the scenes, test subject group and question form is described. The last part of this section describes the outcome of the experiment.

4.3.1 Scenes

Two scenes are used in the experiment where the first one is the base scene and the second one is a variant of the first one. The scenes are shown in figure 12 and are aimed at giving a way to do basic audio localization where the second one adds a step of non-direct audio way to the audio source. A non-direct way to the audio source as in the second scene is aimed to test the case where only reflected sound waves reach the listener.

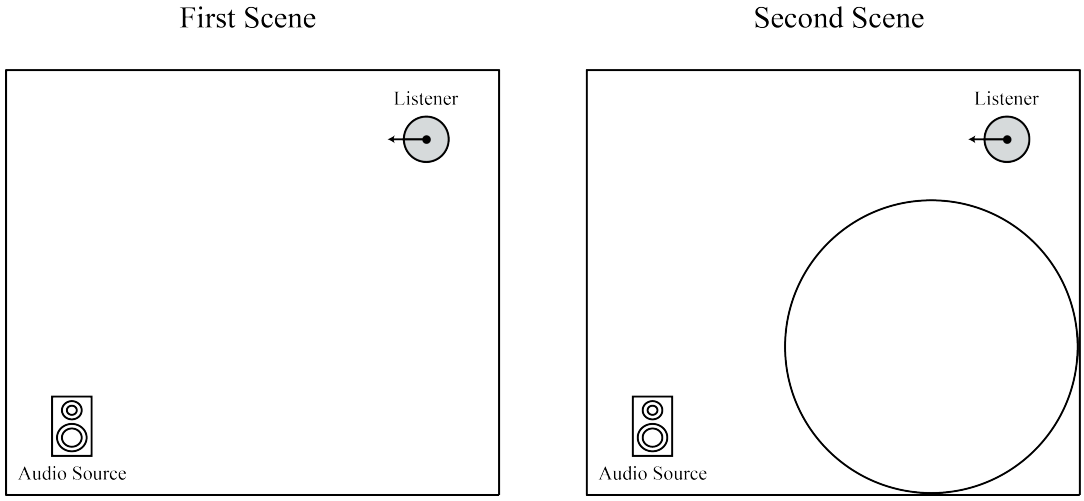

Figure 12. Two scenes are used in the audio experiment where the second one is a variant of the first one; the scenes are shown as slices in the X-Z plane.

4.3.2 Test Subject Group

In this test an age group that would hold a high amount of experience of digital games and virtual environments is used, pupils at a high school in the age group of 17-19 is assumed to have this level of experience of digital games. The education program selected for this experiment is the media studies program in which the pupils is assumed to be naturally accustomed to computers in general and virtual environments in particular.

4.3.3 Question Form

The first section of the question form handles the basic information about the test subject that consists of age, sex and experience with digital games. The experience of digital games is here defined as playing digital games regurarly either on a mobile or desktop platform including mobile phones, gaming consoles and computers. Playing regurarly is here defined as playing games at least two occasions per month on any of the available platforms.

The second section of the form handles the actual questions to the test subject, as there are two different types of sounds involved in the experiment the questions are repeated once with a different sound. With these questions the two sounds were a female voice speaking Swedish with a frequency content mainly below 1000 Hz and a sine wave with a constant frequency of 700 Hz. The realisation of the questions connected to the method was the following:

1. The sound is playing in the first scene, the visual representation of the audio source is not shown. Question: Please give a direction of the audio source?
2. The sound is playing in the second scene, the visual representation of the audio is not shown. Question: Can you tell the difference of the audio source position?
3. The sound is playing in the first scene, the visual representation of the audio source is shown. Question: Does the direction of the audio source match what you are hearing?
4. The sound is playing in the second scene, the visual representation of the audio source is shown. Question: Does the change match what you are hearing?

The first half of the form aims to find the localization possibility without any visual guidance from the test subject. When changing the scene this should not affect the localization of the audio source if the simulation is valid. The second half of the form aims to find the localization possibility by reversing the situation. The reversed situation is based on showing the audio source and asking the test subject to confirm the position.

4.3.4 Outcome

The outcome from the experiment is the collected answers from the test subjects to the questions above. Answers are noted in a basic manner where only the localization direction is noted even if the test subject talks about other topics connected to the experiment. Only the direction is noted as only the basic localization possibility is interesting to examine here.

From the collected answers each question is compared between the test subjects to get a summed answer on that particular question. To get a summed answer each answer is compared to the correct direction where the correct direction is based on the current scene. The summed answer indicates whether a direction is possible to determine for the test subject group and indicates if a basic audio localization is possible.

The test subject group consisted of 7 persons in the age span of 17-19 years with the sex distribution of 3 females and 4 males. As described above the test subject group consisted of high school students at the media studies program. In this group the digital game experience as defined in the overview section were divided as 7 subjects with experience and 0 subject without experience. The experience of people at this age group was at the expected level as assumed in the overview section.

For the experiment the mobile hardware setup was used that is described in the performance section above. The choice of hardware setup does not affect the experiment as only the time consumption differs between the two setups, the audio output is identical when the same configuration of variables in the tracing is used.

The summed answers show that a fine-tuned localization of the audio source was not possible when the audio source position was not known. This conclusion is based on the fact that more than half of the test subjects were able to give a coarse but correct direction but the other test subjects were not. The position of the audio source was not

known in the first half of the question form where the audio source was not visually rendered.

When the position of audio source was revealed in the latter half of the question form the summed answers show that the test subjects were able to confirm this position of the audio source. This conclusion is based on that all but one test subject confirmed the position of the audio when asked. It should be noted that exactly the same position was used in all questions even though the test subjects did not know this.

The aim of the complementary experiment was to get an indication of the validity of the audio output of the simulation. As noted above this was possible to a certain degree but an exact localization was not possible. This result indicates that the basic method of simulating audio is usable but may require some more simulated audio phenomena.

4.4 Analysis

In this implementation only a basic simulation system was implemented, as the intention is to examine the feasibility of real-time usage. The basic simulation uses reflection on all surfaces and computes the contribution to the left and right channel at the listener position. At the reflections no energy loss or altered frequency content is computed (Naef et al., 2002), the sound waves are also reflected indefinitely only limited by the maximum number of iterations in the tracing. The transfer function from sound waves that affect the listener to the left and right channels use a simple method that are described above, a more complex transfer function that simulates head-related phenomena is not part of the implementation.

A sphere tracing based audio simulation gives a straightforward way of adding new simulation phenomena inherited from the ray tracing method. All iterations for each sound wave are traced, which enables phenomena to be implemented with a high level of detail and control.

The performance tests show that this audio simulation system is usable as a part of a real-time environment as the performance cost for the system allows a parallell interactive graphics renderer to run at an interactive frame rate. This is only possible if the audio simulation can run in real-time but not consume all resources of the GPU in terms of processing. As discussed above the ratio between audio and graphics in most interactive environments favor graphics to a high degree. This system is usable at an interactive frame rate and gives a performance window left for the graphics, with a likely lowest frame rate allowed of 30 frames per second each frame must take less than 33 milliseconds to render. The sphere tracing simulation is able to use a smaller part of this time slice and still be able to output audio.

As this audio simulation system is shown to work for real-time usage when implemented it is logical to assume that the sphere tracing method is in at least one form feasible for audio simulation in real-time on the GPU. The parallelized nature of the GPU matches the computations necessary in this simulation well and so can be utilized to a high degree in terms of performance.

With the complementary audio localization experiment this audio simulation system showed an important finding. The audio output works at a coarse level when humans try to localize the audio source but a clear positioning was not possible. The simple

sound wave simulation that is used in this implementation can explain this fact, more simulated phenomena is assumed to give better results in this area.

As Hart (1996) proposed some enhancements like bounding volume hierarchy and space partitioning methods would increase the performance in the system, as the sphere tracing computations would be more efficient. An implementation of these enhancements would also enable more complex geometry to be used without less impact on the performance.

The performance tests also show that hardware implementation details differ between GPU units. The amount of sound waves in the system is based on the resolution of the textures and in the last measurement of each test the mobile hardware setup was not able to handle the amount of data, this gave an unreliable timing value. Under normal conditions the mobile hardware setup was able to run the system with less time consumption than the desktop hardware setup, this is assumed to stem from the fact that the GPU used in the mobile setup is from a newer hardware generation.

A more complex scene gives a less clear path between the audio source and the listener, this is confirmed by the tests with the obstacles scene. Only a small amount of the traced sound waves reach the listener and so the majority of waves are useless for the audio output. It is not trivial to cull these sound waves at the audio source, a starting point for such a method could be based on what waves have the most clear path to the listener.

5 Conclusion

Digital games are interactive virtual environments that use a combination of graphics and audio to immerse to user in the environment presented. The audio is generally underdeveloped compared to the graphics. Virtual environments like digital games can also be solely based on an audio presentation where no graphics is used at all, this type of game is suitable for people with visual impairments. An environment with or without graphics needs enough developed audio enough to make a worthwhile experience. Graphics and audio in environments that require a high level of emulated reality should be combined on a similar level and match in levels of realism. To make the audio sound in a convincing and realistic way a system for simulating audio is needed.

To be interactive a game environment runs in real-time, this fact puts a time constraint on the rendering of the graphics and audio. The time constraint is based on the time window to render on frame in the synchronized rendering of graphics and audio based on the frame rate of the graphics. Consumer computer systems of today are in general equipped with a GPU, if an audio simulation can use the GPU in real-time this is a possible implementation target in a game system.

Previous attempts at using ray tracing methods for audio simulation were successful and this work uses ray tracing as the base method of simulation. With this method sound waves are traced from the audio source to the listener in the virtual scene. A key part of the tracing computations is finding the intersection point between a sound wave and the surfaces in the scene. Sphere tracing is an alternative method for finding the intersection point in the tracing computations and has been shown to be feasible for real-time usage on the GPU. In this case implicit surfaces are used in the scene with the sphere tracing method.

This thesis shows that audio simulation with implicit surfaces using sphere tracing is possible to use in real-time with the GPU in some form. The parallelized nature of the GPU suit the sphere tracing computations in terms of computation time, this should be compared to a similar implementation in the CPU. The time consumption for an audio simulation system like this is small enough to enable it for real-time usage. Based on an interactive graphics frame rate in a game system the time consumption allows the graphics and the audio computations to use the GPU in the same frame time.

The audio output of the implementation in this thesis is emulating audio to some level as localization of the audio source was possible to some degree in an experiement. Humans were able to localize the audio source in a basic manner given a visual representation of the scene, however localization to a finer degree was not possible. As this implementation is intended to examine the feasibility for real-time usage it lacks simulation of audio phenomena that might improve the audio output in terms of localization. An improved audio output would gain a higher level of emulated reality of the audio that matches the virtual environment to a higher degree.

6 Future Work

As argued in the conclusion chapter some elements are lacking in the audio simulation implementation to make it possible to render an audio output with a high level of emulated reality. For a higher level of emulated reality an audio localization should be possible to a fine degree. This fact was shown by the audio localization experiment where a basic localization was possible but not with pinpoint accuracy. In order to enhance the audio simulation more phenomena need to be simulated.

As this sphere tracing based audio simulation is in at least one form possible to run in real-time it is possible to simulate more audio phenomena but still adhere the time contraint for real-time rendering. In the simulation system of Naef et al. (2002) computations for sound wave energy and frequency content manipulation were used that is assumed to improve this simulation. These improvements can be implemented in this system followed by a repeated localization experiment in order to examine whether the new audio phenomena emulates realistic audio to a higher degree.

The current design of the implementation uses textures for storing sound wave data as described in the implementation chapter, these textures are reused for each iteration pass so that input data changes place with output data. This solution will render all texels in the textures in all iteration passes, this means that each sound wave will always be processed no matter the status of the wave. If the tracing for a wave has stopped each pass will still need to examine this wave to conclude that nothing will be done regarding computations. Even if less time is spent on the computations of these waves this is a time slot that could be trimmed away in the system to save total computation time. In the later iterations the number of stopped waves also increase in number. A possible future investigation should aim to optimize this aspect of the simulation through an alternative design for the GPU implementation.

The measurements of the system could be expanded in volume for future investigations, in this thesis two different types of scenes is used with a limited amount of implicit surfaces. A detailed investigation of different types of implicit surfaces used in the audio simulation would add new information about scene creation regarding the choice of surfaces.

For both the performance measurements and the localization experiment a future investigation would use a detailed virtual environment with the corresponding simplified scene for the audio simulation, as discussed in the background chapter. In this thesis only simplified scenes are used that represents a shell of a detailed virtual environment, the visual representation was also based on these scenes. The new investigation would use the detailed virtual environment for the visual representation and the simplified scene for the audio simulation.

As argued in the experiments chapter there is at least one more feasible method for verifying the audio output that compares a real recording from a real environment to the simulated audio from a virtual representation of the this environment. Here a method of comparison must be developed, as a sample-by-sample approach is not usable. This method gives a structured way to automate comparisons for measurements and significantly increase the speed of evaluating the audio output, this should be compared to conducting an experiment using humans.

References

Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). *Real-time rendering*. Wellesley: A K Peters, Ltd.

Blauert, J. (1997). *Spatial hearing: The psychophysics of human sound localization*. Cambridge: The MIT Press.

Friberg, J., & Gärdenfors, D. (2004). Audio games: New perspectives on game audio. *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. 148-154.

Funkhouser, T., Carlbom, I., Elko, G., Pingali, G., Sondhi, M., & West, J. (1998). A beam tracing approach to acoustic modeling for interactive virtual environments. *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 21-32.

Glassner, A. S. (1989). *An introduction to ray tracing*. London: Academic Press.

Hart, J. C. (1996). Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, *12*, 527-545.

Knoll, A., Kensler, A., Hansen, C., Hijazi, Y., Schott, M., & Hagen, H. (2007). *Fast and robust ray tracing of general implicits on the GPU*. Technical report. Utah: University of Utah.

Naef, M., Staadt, O., & Gross, M. (2002). Spatialized audio rendering for immersive virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology*, *2002*, 65–72.

Neuhoff, J. G. (Ed.) (2004). *Ecological psychoacoustics*. San Diego: Elsevier Academic Press.

Purcell, T. J., Buck, I., Mark, W. R., & Hanrahan, P. (2002). Ray tracing on programmable graphics hardware. *Proceedings of ACM SIGGRAPH, 21*, 703-712.

Raghuvanshi, N., Lauterbach, C., Chandak, A., Manocha, D., & Lin, M. C. (2007). Real-time sound synthesis and propagation for games. *Communications of the ACM, 50*, 66-73.

Rindel, J. H. (2000). The use of computer modeling in room acoustics. *Journal of vibroengineering*, *3*, 41-72.

Singh, J. M., & Nayaranan, P. J. (2009). Real-time ray-tracing of implicit surfaces on the GPU. *IEEE Transactions on Visualization and Computer Graphics, 27 Mar. 2009*.

Whitted, T. (1980). An improved illumination model for shaded display. *Communications of the ACM, 23*, 343-349.