COMPUTATION OF ROOM ACOUSTICS USING PROGRAMABLE VIDEO HARDWARE

Marcin Jedrzejewski
PJWSTK
marcinj@aster.pl

Krzysztof Marasek
PJWSTK
kmarasek@pjwstk.edu.pl

Abstract This paper describes a new method of generating real time acoustics with the use of widely available graphic video cards. Algorithm that was mapped to video hardware is ray tracing. Computed echogram was later used during real time auralization. Simplified acoustic model allows to use this method in real time simulations like video games or fast acoustics aproximation. Advantage of this method is ability to move source and listener during simulation without the need for long precomputation phase. Test scenes consisted of highly occluded building architectures. Results show that performing acoustic computation on GPU can significantly speed it up. Propagation of 16384 rays (at 10 reflections) for few room flat took about 32ms on AMD 2GHz with ATI Radeon 9800. Such results allow for real time simulations. Computation of above one million rays took around 1s. Described algorithm is optimized for working with ATI Radeon 9800 hardware with the use of DirectX 9 API.

Keywords: acoustics, GPU, video hardware, ray tracing, impulse response

1. Introduction

Acoustics effects are highly desirable in computer simulations. Applications like video games or presentations of architectural environments can gain quite a lot from adding this new medium. This paper is focused on implementation of sound paths propagation on GPU. Method choosen for auralization is quite straightforward. We want to show that todays graphics hardware can be used to do something it wasn't designed for. This is actually a trend that can be seen in many other fields like physics or mathematics [1].

Graphics Programable Unit

Before going to detailed description of used method, we will introduce GPU programming model techniques. The most important fact to realize is that there is a big difference between programming for CPU and for GPU also called streaming processor. CPUs are executing instructions in serial mode while GPUs are working by executing instructions parallely. Streaming model means that there are many processors executing parallely the same code and each of them operate on some input data and produces some output data. Those processors cannot communicate. Code that is executed on the stream processors is also called kernel. Important fact is also that GPUs operate on vectors. Vector is a collection of four floats, in graphics they mostly refer to RGB components and fourth value is used mainly for alpha blending.

Input data that is going to be processed by GPU kernel is first copied to texture surfaces and later its being accessed by Pixel Shader. Pixel shader is a program that is executed on video hardware for each pixel shown on screen. Texture surface is a block of memory that is stored inside graphics card and can be accessed from inside pixel shader program.

Acoustic model

Precision of computed impulse responses highly depend on the needs. Our main purpose was to compute echos of one sound source in highly occluded environment. Each wall contains absorption coefficient for only one frequency. This restriction is imposed by hardware limitation. Doppler shifting and interaural time difference can be added during auralization and is described later. Model also computes distance of the traced path in order to calculate sound attenuation.

Ray tracing method

Ray tracing was widely studied over the last few decades. Recently there was a lot of work put into runing it on graphics hardware. There is a difference between ray tracing for graphics and acoustics purposes. The first one computes visually accurate scenes that mostly are composed of hundreds of triangles that contains their own shading informations. Also graphical ray tracing requires big screen resolutions like 800x600 or even more, while in acoustics resolutions of render target textures range from 128x128 to 256x256 (128x128 means 16384 traced rays). In acoustic modeling rays are traced from source until they reach receiver most often aproximated by bounding sphere, this is ilustrated in Figure 1. Architecture consists only of polygons that represents walls, all detail objects like furniture, lamps or desks can be omited because they contribute to final results in rather small amount. If there is a need to add for example a bed as an object that will damp sound waves its enough to aproximate it with polygon of appropriate absorption cooeficient.

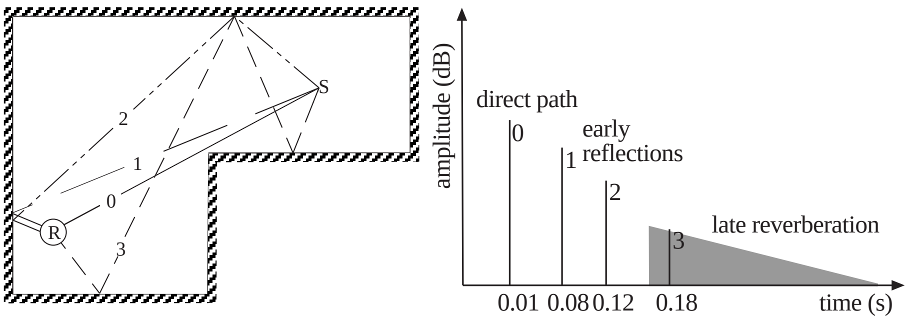

Figure 1. On the left we have a top view of a room with one point source (S) and one receiver (R), there are four traced sound paths of different orders (0,1,2 and 3). Right picture shows computed echogram, each vertical line (also called tap) represents one echo for each traced path.

2. Implementation

Most important aspect of efficient algorithm implementation on GPU is the choice of acceleration data structure. Each ray needs to know with which polygons it should interact at the given moment. The brute force method requires intersecting each ray with each polygon in the scene and chosing the closest one. Computational cost of it is very high. Our method makes use of spatial coherency and is divided into six phases. Instead of computing intersections with polygons we intersect rays with planes which is much more efficient. First four are done during precomputation stage, last two are done in real time.

Phase 1. *Modeling of the scene.* Used algorithms requires input geometry to be "legal". This means that walls are composed of boxes that are combined together. Each wall is made of the set of polygons and each polygons front face (the one from which point normal vector) cannot see back face of any other polygon. In order to make geometry always legal, CSG [1] is computed with exclusive union boolean operator used. This precomputation is made each time geometry changes.

Phase 2. *Leafy BSP[2] tree computation.* This process partitions our geometry into solid convex regions also called leafs. More on algorithm implementation can be found in work by Ranta-Eskola [2].

Phase 3. *Portals computation.* This is done using previously computed BSP tree. Each portal informs to which leaf we can go from current leaf.

Phase 4. *Further leaf splits.* Using planes instead of polygons is difficult in situations where portal and polygon lay on the same plane in the given leaf. In such cases there is a need to make additional leaf splits. Such operation might split other portals which requries path to neighbouring leafs and computation of a new portal information.

Phase 5. *Multi pass rendering.* In each loop of iterations two pixel shaders are executed. First one computes intersections in current leaf for each ray, second one computes propagation to new leaf, intersection with receivers sphere and reflected ray. Because leafs can contain more than 6 planes, ray can stay in current leaf for further plane intersections in the following pass.

Phase 6. *Echogram construction.* It involves retrieval of render target texture with all the final rays and their absorption cooefficients. This is the major bottleneck in our algorithm. Transfering any data from video memory to system takes very long and can take up to 16ms to get 128x128 16bit floating point pixel format texture. This will change in the near future when AGP hardware will be substituted by PCI-Express busses. Based on data computed during ray propagation each image source is calculated. This information is enough to place 3D sound source in space and also use HRTF to model effects like interaural time difference or pinna response. All of this is implemented with the use of the DirectX framework.

Texture data organization

Information on planes, portals and leafs are stored in the form of 1D textures. Each plane/portal have its own information in separate

[1]constructive solid geometry
[2]binary space partition tree

texture (param texture). It contains absorption coefficient and whether its a polygon plane or portal plane. As can be seen in Figure 2 leaf data contains index into plane texture and also count of how many planes it contains (most often six).

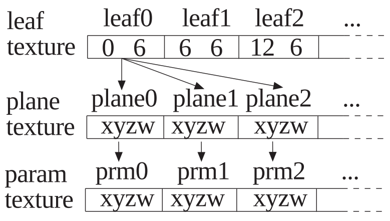

Figure 2. Texture input data organization

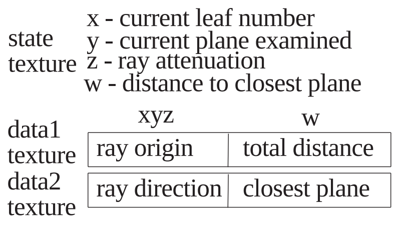

Figure 3. Pixel shader input and output data organization.

Pixel shader operate on render target textures in cyclical way. Organization of data is shown in Figure 3. Three render target textures are used. First one contains state information, second information on origin of ray and third coordinates of ray direction.

Pixel shaders

Pixel shaders were written using HLSL language, they are highly optimized to stay below limit of 96[3] instruction for ATI 9800. Main goal was to compute intersections with six planes during one pass. Pseudocode for both of them is as follows:

```
leafIntersect(ray){
  - Get plane index for current ray
  - Intersect ray with six planes for current leaf
  - Store information of closest of intersected rays
}

rayPropagate(ray){
  - Check if current leaf contains more planes
  - Check if ray is in leaf where listener is located
      If yes then intersect ray with bounding sphere
  - Check if there was intersection with plane
      If yes then reflect ray and its absorption
      If it was portal then set new leaf for this ray
}
```

[3]64 for arithmetic + 32 for texture

Auralization

Real time acoustics generation requires fast methods of signal convolutions. Our method was choosen only for presentation purposes. We assume some maximal number of sound sources and choose from echogram uniformly echos that have different delays. Each delay is computed with the use of distances that rays have traveled. Having this information we can set each sound source to appropriate position to simulate echos.

3. Results

Algorithm shown above was implemented in software without use of GPU. Only difference was to use polygon intersection instead of intersection with all planes in leaf. This way we were able to take benefit of CPU ability to dynamically break `for(;;)` loops. Test case involved tracing 16384 rays uniformly distributed on the source sphere. All rays were propagated until reaching 10 reflections. CPU implementation took almost 0.5 second while GPU version took only 32 ms. Also in those 32ms we have almost 16ms which is time it takes to transfer texture data from video memory to local memory. CPU version of algorithm wasn't very highly optimized. Wald et al. [3] have implemented graphical ray tracer capable of rendering 160 mln. ray-polygon intersections per second using cluster of 7 dual-Pentium III's (800-866MHz). Since we are dealing with acoustic ray tracer its difficult to compare those results. Implementation on GPU was capable of computing over 100mln ray-polygon intersections per second with data retriveal and almost 220mln without[4].

4. Ackowlegments

The authors are grateful to Dr. Maria Tajchert from Warsaw Technical University for the helpful comments and suggestions.

References

[1] General purpose GPU programming site `http://www.gpgpu.org`.

[2] Samuel Ranta-Eskola. Binary space partioning trees and polygon removal in real time 3d rendering. Master's thesis.

[3] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003.

[4]with the incoming PCI-Express architecture this result will be very realistic.