Stage: real-time volume rendering with hardware-accelerated ray casting

Figure 1: Left: Stack of 2D greyscale images from a CT scan of a hand. This stack defines a 3D volume of scalar values. Center: Volume rendering of the CT hand dataset. Right: Volume rendering of a human head.

1 Volume rendering

Ray casting is a popular technique for rendering scalar volumes (see figure 1). It casts hypothetical rays from a specified camera position through each pixel of an on-screen viewport. These rays are then sampled at uniform or variable intervals and in each sample point the volume is interpolated to acquire a scalar value in that point. Transfer functions are used to map the scalars to colors and opacities. For each ray these values are composed in order to determine the color of the pixel.

Ray casting has the advantage that is very flexible and some times faster than other volume rendering methods (e.g. 2D/3D texture mapping) for certain complex volumes where ray traversal can be terminated early. However, in many cases, standard ray casting on high resolutions (many pixels, thus many rays must be cast) and large volumes (long rays, thus many sample points) is too slow for real-time interaction with the volume and transfer functions. The goal of this project is to speed up ray casting using programmable GPUs in order to accomplish such interactivity.

2 Programmable GPUs

During the last years, the performance of GPUs has increased faster than that of CPUs, and for some applications GPUs outperform CPUs. There is also a recent trend in making GPUs more programmable. The newest generations of graphics cards have programmable vertex and fragment (pixel) processors which can be programmed using high-level programming languages such as the OpenGL Shading Language (GLSL), NVidia’s “C for Graphics” (Cg) and Microsoft’s High-Level Shading Language (HLSL) for DirectX. As opposed to earlier, low-level assembler-like languages for programming GPUs, these high-level languages, which are very similar to C, make it easy to implement increasingly complex algorithms to run on GPU [1].
3 Project

Description

For this internship, the possibilities of programmable vertex and fragment processors must be investigated and used to implement a ray-casting algorithm on GPU. The student should do the following:

1. Investigate possibilities of programmable graphics cards and learn to program using high-level shading languages.
2. Investigate existing implementations for GPU-based ray casting.
3. Investigate limitations of existing cards (limitations on volume size, possible dimension of transfer function, complexity of data properties that can be computed on board, etc.)
4. Implement a ray-casting algorithm that runs on GPU. This implementation has several requirements:
   • It must be done in C++.
   • It must be done in OpenGL using GLSL [2][3].
   • It must have a clearly-defined API. This can be accomplished, for example, by implementing it as a VTK [4] class.
   • It must be able to compile the code on both Linux and Windows.
   • The code must be modular such that certain parts of functionality can be easily replaced/enhanced. One particular part of the code that must be flexible is the computation of color and opacity in each sampling point on a cast ray, where the extensions listed below are kept in mind.

Possible extensions

If time permits, one or more of the following extensions can also be implemented:

1. Computation of data properties (e.g. gradient magnitude, LH values [5]) on the graphics card.
2. Support for multi-dimensional transfer functions [6] where color and opacity are computed from more than one scalar value in each point.
3. Computation of color and opacity depending on the camera direction (see also next point).
4. Calculation of tensor properties [7], which may be used in combination with the view vector to compute a view-dependent color and opacity for sampling points.

In any case, the implementation must be flexible and modular enough to allow easy implementation of these features.
4 Practical

Supervisors:

- Petr Šereda (WH 3.2a, 040-247 5571, P.Sereda@tue.nl)
- Tim Peeters (WH 3.2a, 040-247 5571, T.Peeters@tue.nl)
- Anna Vilanova i Bartolí (WH 2.103, 040-247 5416, A.Vilanova@tue.nl)

Start date: 2005
Duration: 3 months
Location: TUE BioMIM lab, WH 3.2a.

Requirements

The student must be a computer science student who should at least:

- Know or learn to program in C++.
- Have experience with (preferably) OpenGL or a similar graphics API such as Microsoft Direct3D, and understand the underlying theory. (E.g. successfully completed classes 2IV7i, i ∈ {0, 1, 2}).

Furthermore, the following is a plus, but not required:

- Experience with (high or low-level) shading languages.
- Experience with using or even implementing classes for VTK.

References


3