Volume rendering
Applications

- Medical visualization
- Scientific visualization
- Computer games
  - Clouds
  - Fire
  - Smoke
  - Volumetric fog
- Visual arts
Volume data
GPU-accelerated ray-casting

• Basic idea:

1. Render the front face of the volume image's bounding box to a 2D RGB texture (starting points)

2. Render the back face of the bounding box to a 2D RGB texture (end points)

3. Subtract the back-face texture from the front-face texture to obtain ray direction (done in the fragment shader)

4. Given the starting points and direction vectors, use the fragment shader to cast a ray from each fragment into the volume image

5. Let the ray gather color and transparency information from the voxels it passes through
Front-face texture
Back-face texture
Ray-casting
Rendering modes

- Through the use of different rendering modes, ray-casting can be used to create a variety of volume visualizations.

- Examples of rendering modes:
  - Maximum intensity projection (MIP)
  - Front-to-back alpha blending
  - Iso-surface rendering
Maximum intensity projection (MIP)

- Basic idea: extract the maximum intensity value along each ray to create an X-ray-like projection of the data
Transfer functions and pseudo-coloring

- Through so-called *transfer functions*, we can map voxel intensity to color and opacity. Example:
Front-to-back alpha blending

• Define a transfer function that maps voxel intensity to color and opacity

• Create a semi-transparent projection of the volume by accumulating opacity values along each ray while composing (blending) colors
Front-to-back alpha blending
Iso-surface rendering

- Extract an iso-surface by selecting the first sample found along each ray that satisfies a pre-defined threshold criteria.
- Compute the gradients (i.e., normals) of the graylevel volume image and use them for shading computations.
Extracted iso-surface
Normals
Shaded iso-surface
Cube environment mapping

- Extract iso-surface and use gradients for cube mapping
Sampling artifacts

- The sampling rate (or step length) affects the visual quality and the frame rate of the volume rendering.

- A low sampling rate enables a high frame rate, but will introduce ugly wood-grain artifacts. Example:
Wood-grain artifacts
How to remove or suppress wood-grain artifacts

• Increase the sampling rate (expensive!)

• Alternative: Hide the artifacts by perturbing the sampling positions with a tiled noise texture (stochastic jittering)
Stochastic jittering

Without jittering

With jittering
Post-processing effects

Original image

Smoothed image
More complicated post-processing effects:

- Glow or light bloom (see GPU Gems, chapter 21)
- Depth-of-field
- Motion blur
- Toon shading

Image courtesy: http://www.realtimerendering.com
Toon shading
Cube environment mapping in GLSL
Procedural noise textures in GLSL

- GLSL implementations of Perlin and Worley noise can be obtained from the following Git repositories:
  - Link 1
  - Link 2 (includes a few demos)
- Just include the noise functions in your vertex or fragment shader
A few words about GPGPU

- **General-Purpose computing on Graphics Processing Units**
- Use the GPU as a Single Program Multiple Data (SPMD/SIMD) parallel processor

- Application areas:
  - Medical imaging
  - Image processing and analysis
  - Physics simulation
  - Molecular dynamics
  - Finance modeling
  - Signal processing
  - Cryptoanalysis (password-cracking!)
GPGPU programming languages

- Compute Unified Device Architecture (CUDA)
  - Created by NVIDIA
  - Proprietary framework
  - Only supported on NVIDIA GPUs :-(

- Open Computing Language (OpenCL)
  - Open standard
  - Managed by the Khronos Group
  - Supported by all GPU vendors (Intel, AMD, and NVIDIA) :-(
  - Enables parallel programming of heterogeneous systems (GPUs, CPUs, and APUs)