[04] GPU-Based Ray-Casting
Talk Outline

- Why use ray-casting instead of slicing?
- Ray-casting of rectilinear (structured) grids
  - Basic approaches on GPUs
  - Basic acceleration methods
  - Object-order empty space skipping
  - Isosurface ray-casting
  - Endoscopic ray-casting
Why Ray-Casting on GPUs?

- Most GPU rendering is object-order (rasterization)
- Image-order is more “CPU-like”
  - Recent fragment shader advances
  - Simpler to implement
  - Very flexible (e.g., adaptive sampling)
  - Correct perspective projection
- Can be implemented in single pass!
- Native 32-bit compositing
Where Is Correct Perspective Needed?

- Entering the volume
- Wide field of view
- Fly-throughs
- Virtual endoscopy
- Integration into perspective scenes, e.g., games
Recent GPU Ray-Casting Approaches

- Rectilinear grids
  - [Krüger and Westermann, 2003]
  - [Röttger et al., 2003]
  - [Green, 2004] (NVIDIA SDK Example)
  - [Stegmaier et al., 2005]
  - [Scharsach et al., 2006]

- Unstructured (tetrahedral) grids
  - [Bernardon, 2004]
Single-Pass Ray-Casting

- Enabled by conditional loops in fragment shaders (Shader Model 3; e.g., GeForce 6800, ATI X1800)
- Substitute multiple passes and early-z testing by single loop and early loop exit
- No compositing buffer: full 32-bit precision!

- NVIDIA example: compute ray intersections with bounding box, march along rays and composite
Basic Ray Setup / Termination

- Two main approaches:
  - Procedural ray/box intersection
    [Röttger et al., 2003], [Green, 2004]
  - Rasterize bounding box
    [Krüger and Westermann, 2003]

- Some possibilities
  - Ray start position and exit check
  - Ray start position and exit position
  - Ray start position and direction vector
Procedural Ray Setup/Termination

- Everything handled in the fragment shader
- Procedural ray / bounding box intersection

- Ray is given by camera position and volume entry position
- Exit criterion needed

- Pro: simple and self-contained
- Con: full load on the fragment shader
Fragment Shader

- Rasterize front faces of volume bounding box
- Texcoords are volume position in [0,1]
- Subtract camera position
- Repeatedly check for exit of bounding box

```c
// Cg fragment shader code for single-pass ray casting
float4 main( VS_OUTPUT IN, float4 TexCoord0 : TEXCOORD0,
            uniform sampler3D SamplerDataVolume,
            uniform sampler1D SamplerTransferFunction,
            uniform float3 camera,
            uniform float stepsize,
            uniform float3 volExtentMin,
            uniform float3 volExtentMax
          ) : COLOR
{
    float4 value;
    float scalar;
    // Initialize accumulated color and opacity
    float4 dst = float4(0,0,0,0);
    // Determine volume entry position
    float3 position = TexCoord0.xyz;
    // Compute ray direction
    float3 direction = TexCoord0.xyz - camera;
    direction = normalize(direction);
    // Loop for ray traversal
    for (int i = 0; i < 200; i++) // Some large number
    {
        // Data access to scalar value in 3D volume texture
        value = tex3D(SamplerDataVolume, position); // Value at this coordinate
        scalar = value.a;
        // Apply transfer function
        float4 src = tex1D(SamplerTransferFunction, scalar);
        // Front-to-back compositing
        dst = (1.0-dst.a) * src + dst;
        // Advance ray position along ray direction
        position = position + direction * stepsize;
        // Ray termination: Test if outside volume ...
        float3 temp1 = sign(position - volExtentMin);
        float3 temp2 = sign(volExtentMax - position);
        float inside = dot(temp1, temp2);
        // ... and exit loop
        if (inside < 3.0)
            break;
    }
    return dst;
}
```
"Image-Based" Ray Setup/Termination

- Rasterize bounding box front faces and back faces [Krüger and Westermann, 2003]
- Ray start position: front faces
- Direction vector: back—front faces

- Independent of projection (orthogonal/perspective)
Standard Ray-Casting Optimizations (1)

Early ray termination

- Isosurfaces: stop when surface hit
- Direct volume rendering: stop when opacity $\geq$ threshold

Several possibilities

- Older GPUs: multi-pass rendering with early-z test
- Shader model 3: break out of ray-casting loop
- Current GPUs: early loop exit not optimal but good
Standard Ray-Casting Optimizations (2)

Empty space skipping
- Skip transparent samples
- Depends on transfer function
- Start casting close to first hit

Several possibilities
- Per-sample check of opacity (expensive)
- Traverse hierarchy (e.g., octree) or regular grid

- These are image-order: what about object-order?
Object-Order Empty Space Skipping (1)

- Modify initial rasterization step

rasterize bounding box  rasterize “tight” bounding geometry
Object-Order Empty Space Skipping (2)

- Store min-max values of volume bricks
- Cull bricks against isovalue or transfer function
- Rasterize front and back faces of active bricks
Object-Order Empty Space Skipping (3)

- Rasterize front and back faces of active min-max bricks
- Start rays on brick front faces
- Terminate when
  - Full opacity reached, or
  - Back face reached
Object-Order Empty Space Skipping (3)

- Rasterize front and back faces of active min-max bricks
- Start rays on brick front faces
- Terminate when
  - Full opacity reached, or
  - Back face reached

- Not all empty space is skipped
Isosurface Ray-Casting

- Isosurfaces/Level Sets
  - scanned data
  - distance fields
  - CSG operations
  - level sets: surface editing, simulation, segmentation, ...
Intersection Refinement (1)

- Fixed number of bisection or binary search steps
- Virtually no impact on performance

- Refine already detected intersection
- Handle problems with small features / at silhouettes with adaptive sampling
Intersection Refinement (2)

without refinement

with refinement

sampling rate 1/5 voxel (no adaptive sampling)
Intersection Refinement (3)

Sampling distance 1.0, 24 fps

Sampling distance 5.0, 66 fps
Deferred Isosurface Shading

- Shading is expensive
  - Gradient computation; conditional execution not free
- Ray-casting step computes only intersection image
Enhancements (1)

- Build on image-based ray setup
- Allow viewpoint inside the volume
- Intersect polygonal geometry
Enhancements (2)

1. Starting position computation
   ⇒ Ray start position image

2. Ray length computation
   ⇒ Ray length image

3. Render polygonal geometry
   ⇒ Modified ray length image

4. Raycasting
   ⇒ Compositing buffer

5. Blending
   ⇒ Final image
Moving Into The Volume (1)

- Near clipping plane clips into front faces
- Fill in holes with near clipping plane
- Can use depth buffer [Scharsach et al., 2006]
Moving Into The Volume (2)

1. Rasterize near clipping plane
   - Disable depth buffer, enable color buffer
   - Rasterize entire near clipping plane

2. Rasterize nearest back faces
   - Enable depth buffer, disable color buffer
   - Rasterize nearest back faces of active bricks

3. Rasterize nearest front faces
   - Enable depth buffer, enable color buffer
   - Rasterize nearest front faces of active bricks
Virtual Endoscopy

- Viewpoint inside the volume with wide field of view
  - E.g.: virtual colonoscopy

- Hybrid isosurface rendering / direct volume rendering
  - E.g.: colon wall and structures behind
Virtual Colonoscopy

- First find isosurface; then continue with DVR
Virtual Colonoscopy

- First find isosurface; then continue with DVR
Hybrid Ray-Casting (1)

- Isosurface rendering
  - Find isosurface first
  - Semi-transparent shading provides surface information

- Additional unshaded DVR
  - Render volume behind the surface with unshaded DVR
  - Isosurface is starting position
  - Start with (1.0-iso_opacity)
Hybrid Ray-Casting (2)

- Hiding sampling artifacts (similar to interleaved sampling, [Heidrich and Keller, 2001])
Conclusions

- GPU ray-casting is an attractive alternative
- Very flexible and easy to implement
- Fragment shader conditionals are very powerful; performance pitfalls very likely to go away
- Mixing image-order and object-order well suited to GPUs (vertex and fragment processing!)
- Deferred shading allows complex filtering and shading at high frame rates
Thank You!

Acknowledgments

- Henning Scharsach, Christian Sigg, Daniel Weiskopf
- VRVis is funded by the Kplus program of the Austrian government