Adaptive Refinement of GPU Volume Ray-Casting

www.vsg3d.com
March 2013
On August 1, 2012, FEI has acquired VSG
VSG: Who we are

Proven track record
- 25+ years of expertise
- 25+ developers
- Highly experienced team
- Successfully serving 1000+ first-class customers

Worldwide company
- USA (Boston, Houston, San Diego)
- France (Bordeaux, Paris)
- Germany (Düsseldorf)
- UK (London)

Distributors worldwide
- Asia/Pacific (China, India, Japan, Taiwan, Korea)
- Mexico, Israel, Russia
Integrate high-performance 3D in your applications

Visualize and analyze scientific and industrial data

Join a partner committed to your success
Solving challenges in demanding markets

**Open Inventor®**
3D Development Toolkit
- Oil & Gas, Geosciences, Mining
- Medical and Life Sciences
- Engineering and Simulation

**Avizo®**
3D Analysis Software
- Materials and Geoscience
- Industrial Inspection
- Engineering & Simulation

**Visilog**
2D Image Processing
- Biology
- Pharmaceutical
- Materials Research

**Amira®**
3D Visualization Software
- Life Sciences
- Biomedical Research
- Pharmaceutical Industry
VolumeViz presentation

OUT-OF-CORE VOLUME RENDERING TOOLKIT
High quality volume ray-casting

- Out of core volume rendering
  - Octree-like hierarchy of resolutions

- Clip, edit and extract data
  - Clip against surface, shapes, masks, ...
  - Edit/extract by voxel, surface, shape, ...

- Customisable shader pipeline
  - GetData: e.g. apply filter
  - CombineData: e.g. apply mask
  - ComputeColor: e.g. blend colors

```cpp
//

vec4 VVizComputeFragmentColor( VVIZ_DATATYPE vox, vec3 voxCoord )
{
  VVIZ_DATATYPE value2 = VVizGetData(data2, voxCoord);
  vec4 color2 = VVizTransferFunction(value2, 1);
  return vox * color2;
}
```
Post-process effects

- Postpone many effects to a deferred pass
  - Deferred lighting
  - Edge detection
  - Ambient occlusion
Traversal effects

- Ray traversal effects
  - Fast cubic BSpline interpolation
  - Volume space self occlusion
  - Multi-data co-blending
  - ...

Image courtesy of ffA
Adaptive screen-space raycasting

A NEW TECHNIQUE FOR ACCELERATING RAYCASTED VOLUME RENDERING
Raycasting optimization techniques

- Decrease ray length
  - Proxy geometry based on tiles’ min/max
  - Early break based on current zBuffer

- Decrease number of steps
  - Boundary binary search
  - Empty space skipping

- Decrease number of rays
  - Low resolution
  - Adaptive screen-space raycasting
Adaptive screen-space raycasting: main idea

- Render in low resolution
- Detect where refinement is needed...
- ... and relaunch rays
- Compose low res pass and high res pass
- Iterate

The process can be halted at any level to maintain interactivity

In practice, 2 or 3 iterations give the best compromise between quality and time
Adaptive screen-space raycasting: detail detection

- Edge detection on red, green, blue
  - Classic sobel filter. Edge detection pass must be fast => keep it simple!
  - Use a threshold to define where rays should be regenerated.
  - Threshold in [0, 1]. 0: everything is edge, 1: nothing is edge.
Adaptive screen-space raycasting: detail detection

- Problem of uniform color map
  - Color is too uniform to correctly detect edges.
  - As a consequence, artifacts appear during deferred pass
Adaptive screen-space raycasting: detail detection

- Edge detection on RGB + depth:
  - Need to linearize Zbuffer:
    \[ z_l = \frac{(2 \times \text{near})}{(\text{far} + \text{near} - z \times (\text{far} - \text{near}))} \]

Depth edge detection retains good quality even in uniformly colored areas.
1\textsuperscript{st} order edge detection

- Edge detection is equivalent to finding 1\textsuperscript{st} derivative.
- Signal (image) is usually composed of steps. Colors change rapidly from one to another:

\[ f, f', f'' \]

\begin{itemize}
  \item 2\textsuperscript{nd} order generates more rays than needed. 1\textsuperscript{st} order works well enough.
\end{itemize}
2\textsuperscript{nd} order edge detection

- However...

- A: Constant signal. Can be low res
- B: Must be full res to keep curvature
- C: Linear part. Can be interpolated from low res
- D: Must be full res to keep curvature
- E: Constant signal. Low res.

- $f$: original signal
- $f'$: edges
- $f''$: edges’ edges
2nd order edge detection

- **However...**

  - **A:** Constant signal. Can be low res
  - **B:** Must be full res to keep curvature
  - **C:** Linear part. Can be interpolated from low res
  - **D:** Must be full res to keep curvature
  - **E:** Constant signal. Low res.

---

This whole area can be interpolated !!!

- \( f \): original signal
- \( f' \): edges
- \( f'' \): edges’ edges
2\textsuperscript{nd} order edge detection

- Especially useful for depth edge detection in flat areas.

1\textsuperscript{st} order: 57,000 rays
2\textsuperscript{nd} order: 16,000 rays

2\textsuperscript{nd} order edge detection is slower than 1\textsuperscript{st} order but generates many fewer rays.

- Cannot always be used, but very efficient in some cases.
Avoid resampling location

- High res and low res samples may correspond. Don’t resample at the same location

  - In odd-sized viewport case, center pixels correspond in low and high resolution.

  - By using a threshold (samples are considered equal if close enough), many samples in center of screen can be reused in full resolution.

  - In practice, this can save about 10% rays.
Avoid resampling location

- High res and low res samples may correspond. Don’t resample at the same location

  - In odd-sized viewport case, center pixels correspond in low and high resolution.
  - By using a threshold (samples are considered equal if close enough), many samples in center of screen can be reused in full resolution.
  - In practice, this can save about 10% rays.
Avoid resampling location

- High res and low res samples may correspond. Don’t resample at the same location

  - In odd-sized viewport case, center pixels correspond in low and high resolution.
  - By using a threshold (samples are considered equal if close enough), many samples in center of screen can be reused in full resolution.
  - In practice, this can save about 10% rays.
Implementation

COMPATIBLE GLSL 1.2 AND OpenGL 2.1

```glsl
float zEntry = 0;
float zExit = 0;
VVizComputeEntryExitPointWithZBuffer(entryPoint, exitPoint, zEntry, zExit);
VVizComputeExitPointWithZBuffer(exitPoint, zExit);

entryPoint.xyz *= VVizVolumeDimensions;
exitPoint.xyz *= VVizVolumeDimensions;

vec3 alignedEntryPoint = floor(entryPoint.xyz + vec3(0.5));
vec3 alignedExitPoint = exitPoint.xyz + (alignedEntryPoint - entryPoint.xyz);

if (entryPoint.w >= 6.0-eps)
{
    entryPoint = VVizAdjustClippedEntryPoint(entryPoint, exitPoint);
}
```
Reminder about rasterization and shader threading

- 8 fragment 2x2 quads are executed as group
  - The whole group has to wait for the slowest thread to finish.
  - Never create a shader with a very long and short branches!

Good case. All the threads are busy.

Bad case. Most of threads do nothing.
Reminder about rasterization and shader threading

- **Early discard using stencil buffer**
  - Stencil test allows early discard and better thread utilization
  - Initialize stencil with edge detection before launching raycast

```
Clear stencil with 1 everywhere
```

```
glStencilFunc (GL_EQUAL, 1);
```

<table>
<thead>
<tr>
<th>Raycast pass</th>
<th>glStencilFunc (GL_EQUAL, 0);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass if stencil is 1</td>
<td>Pass if stencil is 0</td>
</tr>
<tr>
<td>Interpolation pass</td>
<td>Full resolution reached?</td>
</tr>
</tbody>
</table>

- Using early stencil test, performance is up to 3 times faster than naïve thread utilization

```
END
```
Results

BENCHMARKS DONE WITH QUADRO K5000 USING VERY HIGH QUALITY PARAMETERS
CAD dataset:

- Artifacts start to be visible with SNR < 35
  - Edge threshold = 0.3 gives the best compromise.
CAD dataset:

- Ideal case: many uniform areas and well-defined edges
- The main part can be interpolated: only sharp edges need to be full res.
CAD dataset:

- Very efficient with big viewport

Framerate [fps] – eps=0.3

- classic
- adaptive

Viewport size:

- 640*480
- 800*600
- 1024*768
- 1200*1600
- 1200*3200
Molecular dataset:

- More edges than CAD dataset but still efficient.
Seismic dataset:

- Limit of the method.
  - Many varying areas.
  - Edges are not well defined.
  - Lot of areas need to be full res.

\[
\epsilon = 0.3
\]
Limit case:

- Efficient to detect step signal:
  - Classic voxelized rendering
  - Adaptive voxelized rendering

- Cannot detect impulse signal:
To go beyond...

- **Technique is screen space decimation**
- **Does not rely on rendering method**
- **Can be applied whenever rendering a single pixel is expensive (any ray tracing-based method)**