CPU-based rendering with OSPRay

(and with various software rasterizers)

Alex Razoumov
alex.razoumov@westgrid.ca

WestGrid / Compute Canada

copy of these slides and other files at `http://bit.ly/ospraybits`
- will download a ZIP file

Visualization in HPC

Traditional approach:

1. run your large simulation on CPU (or perhaps GPU) nodes
2. do rendering on GPU nodes (if available), or on separate visualization systems, or on personal desktops
   - make use of GPU hardware acceleration through standard graphics APIs (OpenGL)
   - all popular 3D scientific visualization tools are based on OpenGL

However, GPU nodes might not be available or might be busy ...

CPU-based rendering

It is possible to run the entire visualization pipeline on a CPU!

There are two options:

- Software rasterizer. *Mesa* 3D graphics library is an open-source implementation of the OpenGL specification
  - can run on a variety of hardware from GPUs (using vendor-provided low-level device drivers) to CPUs (using various software drivers)
  - of special interest to us are (1) the Gallium *llvmpipe* software rasterizer and (2) the newer *OpenSWR* software rasterizer from Intel
- Software ray tracer. Many implementations. *OSPRay* is a new fast open-source ray tracing engine for Intel CPUs that replaces traditional rasterizers for solid surfaces and volumes

To learn more about the difference between ray tracing and rasterization

`https://youtu.be/DtfEVO9Oc3U` by OnlineMediaTutor

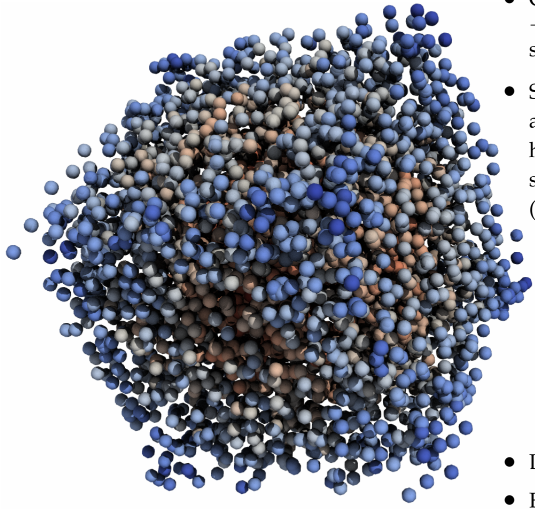
OSPRay `http://www.ospray.org`

- Fast, open-source, end-user ray tracing library from Intel
- Uses Intel's SPMD Program Compiler (ISPC) to properly target different vector instruction set architectures (SSE4, AVX, AVX2, AVX-512)
- Built on top of Embree low-level ray-tracing library (for surface geometry rendering)
  - Embree does not render images by itself, useful for building ray tracers
  - based on recent advances in ray tracing algorithms ⇒ 1.5-6X speedup compared to earlier software renderers
- Unlike GPU, full access to (much larger) host memory
- ParaView, VisIt, VMD already use OSPRay
- Same spirit as OpenGL (both surfaces and volumes), but different API
- For surfaces can work with non-polygonal geometry: cones, spheres, streamlines, cylinders
- Small memory footprint: can handle $10^{8.5}$ triangles, $10^{9.5}$ particles on a single workstation
- It's ray tracing ⇒ provides ambient occlusion and shading, speed depends on your setup
- Can use one of three backend devices
  - LocalDevice for rendering on a single compute node / workstation
  - COIDevice for offloading to first-generation Xeon Phi cards
  - MPIDevice for MPI-parallel rendering (in ParaView's OSPRay?)

OSPRay in ParaView

- First implemented by TACC as a plugin *pvOSPRay* (precompiled Linux binary or source code) starting from ParaView 4.3.1
- Now directly integrated starting from ParaView 5.1
  - currently supports both Linux and MacOS and “experimentally” Windows; included into *precompiled* ParaView 5.1+ on all platforms
  - can switch OpenGL $\longleftrightarrow$ OSPRay at runtime – see the option “Enable OSPRay on/off” in the View -> View panel
  - if compiling ParaView yourself, you’ll need Intel’s SPMD Program Compiler (ispc) and Intel’s Threading Building Blocks (tbb) library, then compile Embree, then compile OSPRay, then compile ParaView with PARAVIEW_USE_OSPRAY=ON – contact me for detailed instructions for Linux

OSPRay in ParaView 5.1+ on a laptop



- On presenter's laptop: `paraview --state=wavelet.pvsm` or build from scratch
- Screen max-resolution bug: in the GUI if attempt to save a screenshot at resolution higher than the maximum for a given screen, get problems in the resulting image (strange interlacing)
    - with and without shadows
    - both in MacOS and Linux
    - with OpenGL no such problem
    - one can avoid it by setting the output image size in a Python script and running it with `pvbatch -use-offscreen-rendering ...`
- Light direction tied to the scene
- Effective in showing the depth effect in complex geometries

Benchmarking: hardware-driver OpenGL vs. OSPRay

- Want to fix the output resolution, have the same camera setup (the object takes equal number of pixels), same everything
- Render 270 frames rotating the object 0.333° between consecutive frames and measure the time it takes to do full 90° rotation
- Assume that saving images takes negligible time comparing to rendering
- The (shortened) code is on the next slide, run the (full) code with either

  ```
  /Applications/ParaView.app/Contents/bin/pvbatch wavelet.py
  /Applications/ParaView.app/Contents/bin/pvbatch --use-offscreen-rendering
  wavelet.py
  ```

  (whether we open a window or not does not affect the time significantly)
- Run benchmark on a single core

Benchmarking: hardware-driver OpenGL vs. OSPRay

```
from paraview.simple import *
import time

wavelet1 = Wavelet()
renderView1 = GetActiveViewOrCreate('RenderView')
renderView1.ViewSize = [1280, 720]
shrink1 = Shrink(Input=wavelet1)
Hide(wavelet1, renderView1)
Hide(shrink1, renderView1)
glyph1 = Glyph(Input=shrink1, GlyphType='Sphere')
glyph1.ScaleFactor = 1.95
glyph1.GlyphType.Radius = 0.2
glyph1.GlyphType.ThetaResolution = 30
glyph1.GlyphType.PhiResolution = 30
renderView1.CameraPosition = [-40.57242525131943, 20.7869159629548, 43.073450283532516]
renderView1.CameraFocalPoint = [0.3880089693718842, -1.4969286758104854, 1.1754828437649194]
renderView1.CameraViewUp = [0.2571900240728232, 0.9264876350141374, -0.27472523306629704]
renderView1.CameraParallelScale = 17.3205080757

# renderView1.EnableOSPRay = 1
# renderView1.Shadows = 1
# renderView1.SamplesPerPixel = 10   # the default is 1

camera = GetActiveCamera()
numberOfFrames = 270
t1 = time.time()
for i in range(numberOfFrames):
    print i
    camera.Azimuth(.333)
    SaveScreenshot('/path/to/frame%04d'%i+'.png', magnification=1, quality=100, view=renderView1)
t2 = time.time()
print t2-t1, 'seconds␣elapsed'
print float(numberOfFrames)/float(t2-t1), 'fps'
```

Benchmarking: hardware-driver OpenGL vs. OSPRay

At 1280×720 resolution

- OpenGL: 3.04 fps (opengl.mp4)
- OSPRay, no shadows, 1 sample per pixel: 2.67 fps
- OSPRay, shadows, 1 sample per pixel: 1.62 fps (ospray2.mp4)
- OSPRay, shadows, 3 samples per pixel: 0.95 fps
- OSPRay, shadows, 10 samples per pixel: 0.39 fps

On my laptop OSPRay is a little slower than OpenGL but not by much if we don't overtax it

At 2560×1440 resolution (4X the number of pixels) ∼4X slowdown for both

- OpenGL's 0.67 fps vs. OSPRay's (shadows, 1 sample pp) 0.43 fps

On a very underpowered 2010 Macbook Air at 1280×720 resolution

- OpenGL's 0.88 fps vs. OSPRay's (shadows, 1 sample pp) 0.21 fps

OSPRay on the cluster

- Similar to MacOS, on Linux recompiled ParaView 5.1+ has OSPRay built-in, so can run on a GPU-less cluster?
- ParaView still requires OpenGL context at runtime (even though you can later switch from OpenGL rasterization to OSPRay ray tracing), so if you try to open it inside VNC, it'll crash without a GPU ...

OSPRay on the cluster

- Fortunately, precompiled ParaView includes Mesa libraries compiled with both *llvmpipe* and *OpenSWR* software rasterizers
  - use `paraview --mesa-llvm` to pick Mesa software rendering with llvmpipe (will work everywhere including older hardware)
  - use `paraview --mesa-swr-avx` or `paraview --mesa-swr-avx2` to pick Mesa software rendering with OpenSWR on processors that support AVX/AVX2 instruction sets

    AVX = Advanced Vector Extensions, support in hardware from ∼2011
- WestGrid's bugaboo cluster's processors are based on the Westmere microarchitecture, so no support for AVX and hence no OpenSWR – instead we'll use *llvmpipe* software rasterizer
  - the rasterizer library `.../ParaView-5.1.2/lib/paraview-5.1/mesa-llvm/libGL.so.1` will be our replacement for the hardware-driver-provided OpenGL

Demo time

Using OSPRay inside ParaView+Mesa+llvmpipe
inside a VNC desktop on bugaboo's compute node

1. On bugaboo.westgrid.ca **submit an interactive job** to the cluster
   ```
   qsub -I -l nodes=b402:ppn=1,pmem=2000mb,walltime=00:60:00
   ```
   When the job starts, it'll return a prompt on the assigned compute node.

2. On the compute node **start the vncserver**
   ```
   vncserver -geometry 1420x820
   ```
   It'll produce *"New desktop is b402:1"*, where the syntax is `nodeName:displayNumber`

3. On your laptop **set up ssh forwarding** to the VNC port:
   ```
   ssh username@bugaboo.westgrid.ca -L 5901:b402:5901
   ```
   Here the second port number 5901 = 5900 (VNC's default) + displayNumber

4. **Start TurboVNC vncviewer** on your desktop, enter `localhost:displayNumber`, e.g. `localhost:1`, and then enter the VNC password if you have set one up. A remote VNC desktop will appear on your screen

5. Back on the compute node **start ParaView**
   `export DISPLAY=b402:1`, where 1 is `displayNumber`
   ```
   module load paraview/5.1.2
   paraview --mesa-llvm
   ```
   ParaView window will appear inside your VNC desktop

CPU-based rendering on the cluster

- Now we are switching between *software-based* OpenGL-like rasterizer and *software-based* OSPRay ray tracing engine, with no GPU in sight
- OSPRay (even with shadows enabled) is much smoother than Mesa+llvmpipe, producing much higher framerates at similar CPU loads

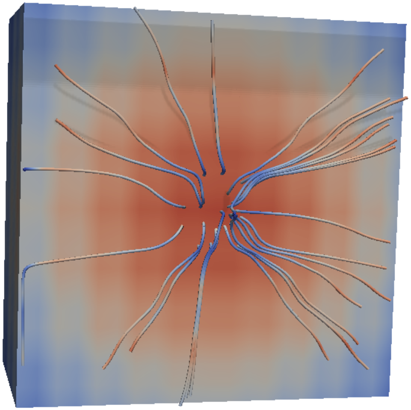
  ```
  module load paraview/5.1.2
  pvbatch --mesa-llvm wavelet.py
  ```

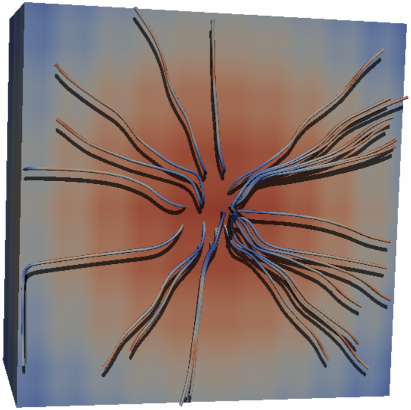
  1280 × 720 resolution

  - Mesa-llvm: 0.144 fps
  - OSPRay, shadows, 1 sample per pixel: 2.61 fps (18X faster!)

Unfortunately, cannot test Mesa-OpenSWR on the same system ...

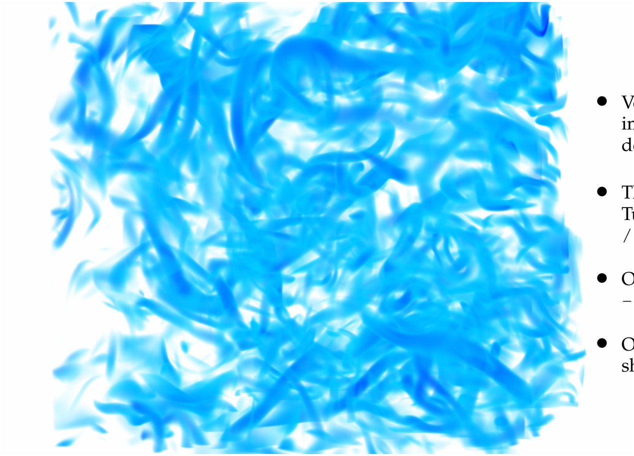
Rendering streamlines





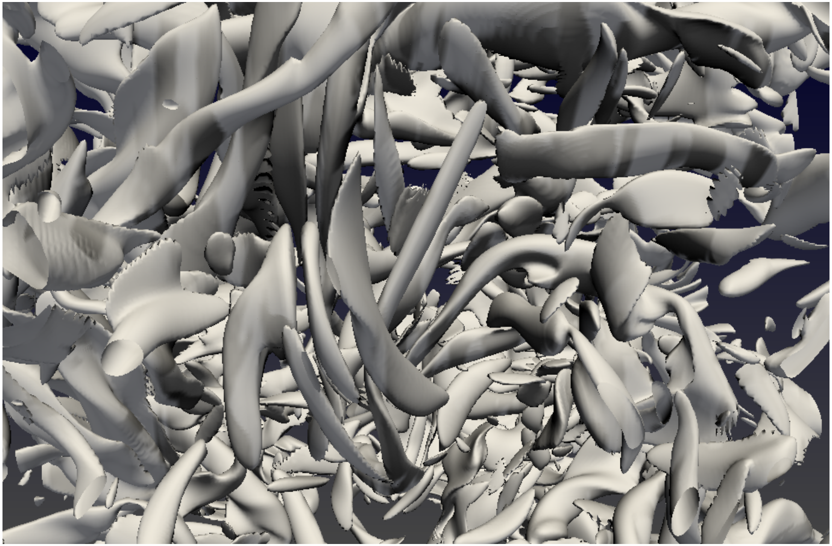
- Cones, spheres, streamlines, cylinders are other types of non-polygonal geometry supported by OSPRay
- Stream Tracer's output in OSPRay works well but the result is not very interesting, unless enabling "OSPRay Use Scale Array" in Properties:Miscellaneous and making a good choice for the scale array
- Stream Tracer + Tube's output in OSPRay (cylinders) is great for showing the depth effect with shadows, but extremely buggy both in MacOS and Linux
- On presenter's laptop: `paraview --state=stream.pvsm`

Volumetric rendering with OSPRay

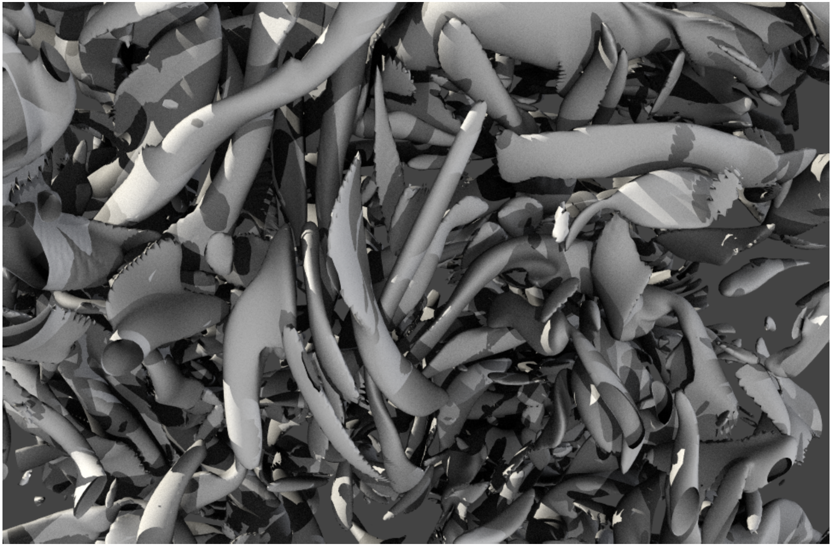


- Volume rendering in OSPRay implemented from scratch (Embree does only surfaces)
- The dataset is 256^3 “TACC Isotropic Turbulence” from `http://www.ospray.org/demos.html`
- On presenter’s laptop: `paraview --state=turbulence.pvsm`
- OpenGL’s 1.96 fps vs. OSPRay’s (no shadows) 1.76 fps

256^3 turbulence: isosurface rendering with OpenGL



256^3 turbulence: isosurface rendering with OSPRay



Conclusions

- OSPRay + shadows is very good to highlight three-dimensionality and depth effect in intricate scenes
- OSPRay is much faster than Mesa-llvm, almost as fast as OpenGL – great for systems without a GPU
- OSPRay works for both surface and volume rendering
- Already included in precompiled ParaView 5.1+ binaries
    - currently supports both Linux and MacOS
    - “experimental” in Windows
- Current implementation is somewhat buggy but this should improve
- Not clear if the current implementation in ParaView supports OSPRay’s MPIDevice (all my tests show that it does not) – will keep investigating
- I have not mentioned client-server OSPRay rendering
- Have not yet benchmarked OpenSWR

Questions?