ParaView and VTK with OSPRay and OpenSWR

David E. DeMarle, Kitware Inc.
Scientific Visualization

- Spatially embedded data (x,y,z)
- Manipulate, Display and Interact with data to gain insight
- For feature discovery
- For communication of results
VTK, ParaView and Catalyst

- **VTK** - open source library for scientific visualization

<table>
<thead>
<tr>
<th>VTK rel</th>
<th>6.0</th>
<th>6.1</th>
<th>6.2</th>
<th>6.3</th>
<th>7.0</th>
<th>7.1</th>
<th>8.0</th>
<th>8.1</th>
</tr>
</thead>
<tbody>
<tr>
<td># authors</td>
<td>73</td>
<td>65</td>
<td>75</td>
<td>52</td>
<td>83</td>
<td>95</td>
<td>105</td>
<td>~68</td>
</tr>
</tbody>
</table>

- **ParaView** - an MPI scalable application framework from VTK

- **Catalyst** - in-situ ParaView (embedded into simulation codes)

\[
\rho \frac{D\vec{v}}{Dt} = -\nabla p + \mu \nabla^2 \vec{v}
\]

Simulation Code + ParaView Catalyst
OpenSWR

- Just another OpenGL?: OpenGL.Mesa.OpenSWR
- Threaded and vectorized, 10..80x faster than llvmpipe
- Regularly tested and passes all of VTK’s 1800+ tests
- 2016 Trinity benchmark 1.1 trillion triangles

Note: Only 1/19th machine. Expect 10-20 trillion tris and about 1 minute per frame at pre KNL max.
OSPRay

- library for ray tracing, not rasterization
- Threaded and vectorized

A1: Pretty

Why Ray Tracing?

ParaView benchmark frame times, 30 million tris, 400x400 images

<table>
<thead>
<tr>
<th>renderer</th>
<th>Haswell</th>
<th>KNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>llvmpipe (5/10/17)</td>
<td><strong>5.291861</strong></td>
<td>18.894011</td>
</tr>
<tr>
<td>OpenSWR (5/11/17)</td>
<td>0.085990</td>
<td><strong>0.309664</strong></td>
</tr>
<tr>
<td>OSPRay (5/11/17)</td>
<td>0.006520</td>
<td><strong>0.042851</strong></td>
</tr>
</tbody>
</table>

A2: Fast!
Architecture of OSPRay in VTK/ParaView

first robust runtime alternative to OpenGL for VTK
more complete separation of state and implementation

#include "vtkOSPRayPass.h"
...
vtkOSPRayPass* osprayPass = vtkOSPRayPass::New();
...
if (useOSPRay) //OSPRay
   renderer->SetPass(osprayPass);
else //GL
   renderer->SetPass(nullptr);

<table>
<thead>
<tr>
<th>PV release</th>
<th>5.0</th>
<th>5.1</th>
<th>5.2</th>
<th>5.3</th>
<th>5.4</th>
<th>5.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ features</td>
<td>prototype</td>
<td>Surfaces, mac/win, hotswap</td>
<td>Regular volumes</td>
<td>Progressive</td>
<td>Stereo</td>
<td>Path tracer, AMR and unstructured Volumes</td>
</tr>
</tbody>
</table>
OSPRay’s scivis renderer (PV 5.0-5.4+)

- Primarily* a ray caster (color of first hit)
  - Output is much like OpenGL: local illumination
  - Layering and depth compositing work exactly as GL

- A few nice *exceptions and benefits over GL
  - *Subsequent frame rate fast
    and logarithmic wrt/ mesh resolution
  - Opacity
  - Hard shadows
  - Ambient Occlusion
  - Implicit sphere and cylinders
# KNL Rendering first results (2016 Q1)

1 KNL node (256 ht cores, 1.6GHz), 94GB

<table>
<thead>
<tr>
<th>mtris</th>
<th>llvm (frame/sec)</th>
<th>swr-avx2 (frame/sec)</th>
<th>OSPRay (frame/sec)</th>
<th>llvm (frame/sec)</th>
<th>swr-avx2 (frame/sec)</th>
<th>OSPRay (frame/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.84</td>
<td>9.57</td>
<td>14.96</td>
<td>0.76</td>
<td>6.24</td>
<td>8.19</td>
</tr>
<tr>
<td>10</td>
<td>0.12</td>
<td>4.92</td>
<td>15.25</td>
<td>0.11</td>
<td>3.80</td>
<td>8.07</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>2.84</td>
<td>15.04</td>
<td>0.06</td>
<td>2.10</td>
<td>7.96</td>
</tr>
<tr>
<td>40</td>
<td>0.06</td>
<td>1.75</td>
<td>14.76</td>
<td>1.39</td>
<td>8.12</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0.81</td>
<td>14.95</td>
<td>7.87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>0.46</td>
<td>14.80</td>
<td>7.77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>0.36</td>
<td>14.58</td>
<td>7.69</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

f₀ = 32sec

f₀ = 71sec
GL points (L) and sprites (C) lack the meso-scale clues that pOSPRay’s (R) ambient occlusion provides. Crack propagation data thanks Souchin Deng @ INL
OSPRay’s pathtraced renderer (PV 5.5+)

- Enabled OSPRay’s photorealistic path tracer in VTK and ParaView
  - more fully simulates complete set of paths that photons take in reality
  - reflections, refractions, soft shadows, caustics: generally global illumination
- Changes to VTK/ParaView
  - Progressive Refinement -
    - because more expensive to compute
    - refinement keeps user interaction comfortable
  - Expanded VTK’s light model and PV interface
    - Ambient and area lights, + luminous objects
  - Added generic material infrastructure
    - Can specify material per actor, block and cell
    - Materials can include textures
    - Open ended infrastructure for future proofing
The new **Light Inspector** panel
Lets you dynamically add, place, edit and delete lights.

Ambient type and Radius are new to VTK and built for OSPRAY

Far more expressive than the standard LightKit + optional headlight we’ve had since PV 1.0 (i.e. either 5 or 6 camera relative directional lights)
Materials are open ended

Json file to describe a collection of:

```
{
  "family" : "OSPRay",
  "version" : "0.0",
  "materials" : {
    "Copper" : {
      "type" : "Metal",
      "doubles" : {
        "reflectance" : [0.7843, 0.4588, 0.2],
        "roughness" : [0.0]}
    },
   "Wood" : {
      "type" : "OBJMaterial",
      "textures" : {
        "map_kd" : "wood.jpg"}
    },
   "CheckerClear" : {
      "type" : "OBJMaterial",
      "textures" : {
        "map_d" : "checker.png"}
    }
  }
}
```

Materials are open ended
Load one or more into ParaVlew via File -> Load OSPRay Materials …
Then we share amongst all processors

```
vtkOSPRayMaterialLibrary (singleton like)

bool LoadFromFile(FileName)
bool Serialize()  //for client/server sync
bool Unserialize()
vector<String> Materials
vtkTexture* GetTextureForMaterial(mat, choice)
double** GetParameterForMaterial(mat, choice)
```

indexed color maps to choose material

```
vtkLookupTable
vtkOSPRayMapper::Render() {
...
if (cell->AnnotatedValue() in MatList):
tell OSPRay to use it
...
}
```
Increase rendering quality as needed

OpenGL
Increase rendering quality as needed

OSPRay
SciVis
Increase rendering quality as needed

OSPRay PathTracer

“for communication of results”
Still have all the capability of ParaView

Data Inspection and Manipulation

“for feature discovery”
• Threshold
• Translate
• Cut
• ColorMap
...
Future Work - there’s plenty of cool things to do!

- More applications
  - Preprocessing (ex. CMB)
  - Medical (ex. 3D Slicer)
- More deployments
- More rendering modes
  - Material combiners
  - Attribute data inputs to materials
  - High dynamic range lighting
- Non-photorealistic materials
  - GL/SWR can reuse the new VTK/PV infrastructure too: just add shaders
- Distributed memory secondary rays for arbitrary scenes
  - Now that path tracer is available - HPC customers will demand it.
- About those texture coordinates …
- Path tracer volume rendering
- Meshless rendering
  - Why ray trace? A3: “different” isovolumes, non-linear elements, …
- Embree enabled filters
  - Collision detection, radiance estimation, …
Thanks

• Come see ParaView/OSPRay demos at Intel booth!
• Come talk to me at Kitware booth (Monday 6-9, Tues 10-12 and generally around SC17)
• Learn all about ParaView at The ParaView Tutorial, Sunday at SC17

• Learn More:
  – https://www.VTK.org/
  – https://www.paraview.org/
  – https://www.paraview.org/Wiki/ParaView/Catalyst/Overview
  – http://www.computationalmodelbuilder.org/
  – https://www.slicer.org/
  – http://www.ospray.org/
  – http://openswr.org/
  – dave.demarle@kitware.com