A GPU-Accelerated Node Based Framework for Hair Simulation and Rendering

Francesco Giordana
Sarah Macdonald
Gianluca Vatinno

Double Negative VFX
Hair

- Creatures:
  - Digi-doubles hair / facial hair
    (100k - 150k)
  - Digital creatures fur and feathers
    (few Ms)

- Environments:
  - Grass, moss, seaweed, etc..
    (many Ms)
Furball

- Procedural node-graph
- Hybrid CPU/GPU
- Real-time previews
- Modular
- Standalone + embedded

Historical first render with Furball 1.0
Furball Framework

C++

Python

GPU Accelerated

Tools integration

Maya
PRMan
Houdini
What language / libraries?

**Thrust:**

- **Fast and easy** to use: **STL**-style with fancy iterators
- Can handle host code: makes code reuse easier
- **CUDA** backend + CPU backends (**TBB** and **OpenMP**)  
- **Problem**: no streams, no manual control of shared mem  
- **Solution**: prototype in **thrust**, optimize later
FurShop - Maya Integration

Real-time preview in Maya viewport

Embedded in Maya Dependency Graph

Custom Graph Editor
FurShop - Tools

- Mask painting tool
- Interactive brush tool
- Attribute publishing
- Custom UI elements
FurShop - Example Workflow

Static geometry

Density mask

Follicles

Hairs

Guides

Final
FurShop - Maya Nodes

FurConversionNode

FurNetwork

FurCache

FurAttributePtr

FurNetwork

FurRenderNode

FurSystem

FurNode

Merged computation chain

MPxData

MPxNode

GPU Technology Conference 2013
dnSynapse

- DAG with lazy-pull computation
- Data flows through Attributes
- Nodes are operators
- Sub-graphs
Double negative visual effects

GPU Technology Conference 2013

dnSynapse - Device Controller

- Initialize and select device
- Create CUDA Context
- Handle resources (e.g. available memory)
- Enable / disable GPU acceleration

```
struct DeviceController
{
  void enableGPU( bool enable );
  void isEnabledGPU();
  void selectBestDevice();
  bool canHandle( const DataGPU* data );
}
```
dnSynapse - Dual Data

- Abstract Data wrapper with interface exposed to user
- Two separate implementations for CPU and GPU
- Data conversion triggered with `getDataGPU()` or `getDataCPU()`

```cpp
struct DataCPU
{
    thrust::host_vector<...> ...
    void clear();
    void save( char* filename );
    void load( char* filename );
};

struct DataGPU
{
    thrust::device_vector<...> ...
    void clear();
    void copyTo( DataCPU* dst );
    void copyFrom( const DataCPU* src );
};

struct Data
{
    DataCPU* dataCPU;
    DataGPU* dataGPU;
    void clear();
    ...
    DataCPU* getDataCPU();
    DataGPU* getDataGPU();
};
```
Nodes have a **CPU compute** and a **GPU compute** (optional)

- Try **GPU compute** first, fallback to **CPU compute**
- Calling compute will make sure data is in the right place (H or D)
- Can enable / disable GPU computation with flags (for debugging)

```c
void compute( Data* outData, Context* inContext )
{
    bool result_CUDA = false;

    if ( cudaEnabled() && canUseCUDA() )
        result_CUDA = computeCUDA( outData->getDataGPU(), context );

    if ( ! result_CUDA )
        computeCPU( outData->getDataCPU(), context );
}
```
Furball - Hair

- **Follicles**
  - Surface Patch ID
  - Surface Patch ST
  - Follicle Position
  - Follicle Orient
  - Follicle UV

- **Curves**
  - $n$ Curve Points
Main families of operators

- **Per-point:**
  - Each point in a separate thread

- **Per-curve:**
  - Compute a whole curve in walking down from root to tip

- **One-curve-to-many:**
  - Relationships between one curve and a set of curves

- **Many-curves-to-many:**
  - Relationships between all curves in a set
Splitting the domain

- **Problem**: Some data-sets are too large or unbalanced
- **Solution**: Split per patch

- **Need**: Fast insertion / removal of curves into a patch
- **Need**: Easily identify and manipulate whole curves

Hair split into two independent subsets (different bounding boxes)
Memory Layout

- Follicles sorted per-patch, each element in separate array
- Curves sorted as follicles, points stored root-to-tip
- Can split components to separate arrays to maximize memory access efficiency
Coalesced Access

- Per-point kernels have coalesced access
- Per-curve kernels have strided access (need to walk along the curve)
- Try to rearrange inputs to achieve per-point kernels
- If strided access cannot be avoided: disable L1 cache, use shared memory
The importance of coalesced access

- Test: rescale 1M curves, 32 points per curve
  - **per-point**: scale each point separately **36ms**
  - **per-curve**: iterate over curve points and rescale each one **96ms**

```cpp
Point operator()(const Point& p, float l)
{
    return p * l;
}
```

**Per-point operator**

```cpp
void operator()(const thrust::tuple<size_t, float>& t)
{
    const size_t idx = thrust::get<0>(t);
    const float l    = thrust::get<1>(t);
    Point* p = &m_points[idx*cNumPoints];
    for (int i=0; i<cNumPoints; ++i)
    {
        *p++ *= l;
    }
}
```

**Per-curve (strided) operator**
Caching

• **Problem:** Caching occupies memory resources and transfer H->D is slow

• **Solution:** Cache follicles, generate hairs on the fly

**Test:** 1 million curves, 32 segments per curve:

• **Follicles and hairs on host, non-pinned memory**
  • Size: **420MB**, H->D: **120ms**

• **Follicles on host, hairs on device**
  • Size: **50MB**, H->D: **10ms**, Hair Generation: **14ms**
Point Compression

- Pack 1 \texttt{float3} vector into 1 \texttt{float2} (containing 4 halves)
- Compressed representation on host must match the representation on device
- Use \texttt{IEEE half}: matches \texttt{__float2half} and \texttt{__half2float} intrinsics
Fragmentation of computation

- Many small operators one after the other
- Always read from global memory to registers, then write out, then read again etc
- Should try kernel fusion
- DSL can be a solution
Test Computer

Mirrors current artists’ computers:

Xeon X5690 @ 3.47 GHz
6 Cores
48 MB RAM
Quadro 4000

CPU - Single threaded using STL containers
CUDA - compute 2.0, using thrust

Soon to test multi-threaded CPU and CUDA on K20
Wisps

Inputs:
- Hairs
- Wisps center curves
- Envelope profiles
- Masks
- Randomization

Steps:
1) Generate envelope for each wisp
2) Distance computation hair follicle - wisp root
3) Randomly pick one of the overlapping wisps for each hair
4) Parallel transport of distance vector along the curve
5) Rescale vector so that it fits the envelope
Wisps

- CPU kdtree vs. CUDA brute force
- 10x speedup on GPU

<table>
<thead>
<tr>
<th>Num hairs</th>
<th>100 Wisps</th>
<th>10k Wisps</th>
</tr>
</thead>
<tbody>
<tr>
<td>50k</td>
<td>CPU-60seg: 210</td>
<td>CPU-60seg: 530</td>
</tr>
<tr>
<td></td>
<td>CPU-30seg: 118</td>
<td>CPU-30seg: 30</td>
</tr>
<tr>
<td>200k</td>
<td>CUDA-60seg: 485</td>
<td>CUDA-60seg: 142</td>
</tr>
<tr>
<td></td>
<td>CUDA-30seg: 78</td>
<td>CUDA-30seg: 125</td>
</tr>
</tbody>
</table>

CPU-60seg vs. CUDA-60seg: 10x speedup on GPU

GPU Technology Conference 2013
Filter Frizz

Inputs:
- Hairs
- Ramp
- Mask
- Randomization

Steps:
1) Generate random sequence per-hair
2) Generate random sequence per-point
3) Premultiply mask value, ramp value and random sequences
4) Modify curves per-point
FilterFrizz

- **3-4x** speedup on GPU
- Complex issue: generate random sequences

![Graph showing performance comparison between CPU and CUDA for different num hairs and time segments.](image-url)
Compute BBox

- Reduction
- Strided access to accumulate point positions
- **5-6x** speedup on GPU
Future Work

- Switch to Kepler
- GPU k-d trees
- Multiple streams
- Multi-threaded CPU
- Compile portions of graph
- Kernel fusion
Questions?

Francesco Giordana - fg@dneg.com
Sarah Mcdonald - svm@dneg.com
Gianluca Vatinno - gv@dneg.com