SIDE EFFECTS SOFTWARE

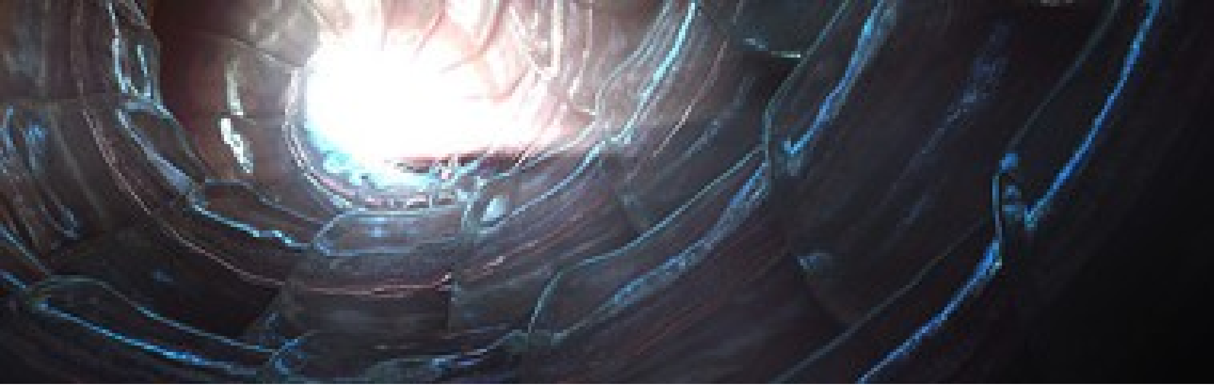
GO | PROCEDURAL

Houdini 12 Development Kit

Houdini
3D ANIMATION TOOLS

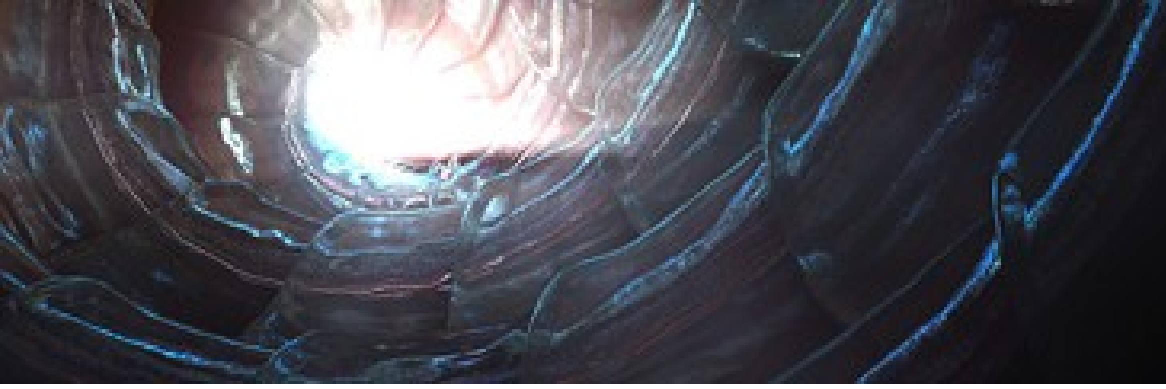
HDK Resources

- Doxygen & HDK Documentation
  - http://www.sidefx.com/docs/hdk12.0/ (or /hdk11.1)
- Locations
  - $HFS/toolkit/
    - samples, include, makefiles
- Specifications
  - $HFS/houdini/public/
    - hgeo, binary_json, GPD, etc.
    - SIM_SolverODE, SIM_SolverBullet



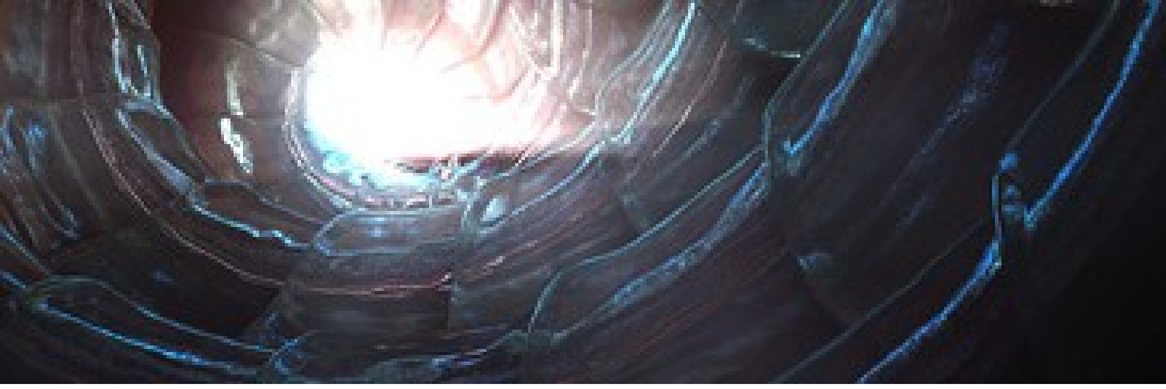
HDK Library Structure

- Hierarchical Set of Libraries
  - Utility Libraries
  - Node Libraries
  - Geometry Libraries
  - Simulation/DOPs
  - Specialized Node Libraries (eg. SOP)
  - Some UI
  - Python Customization



Utility Libraries

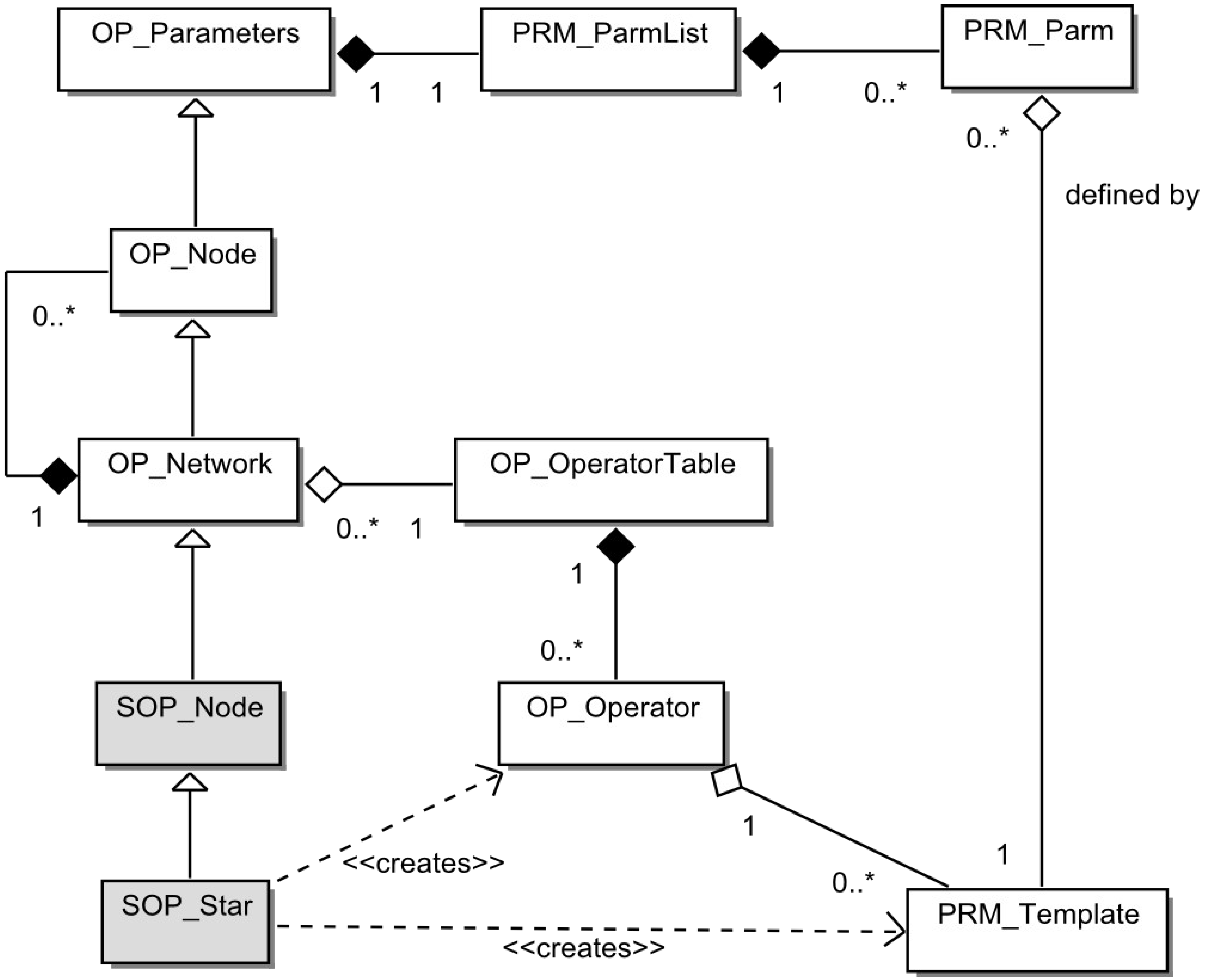
- AU – Audio
- CL – Channel Clips
- CVEX – C++ interface on VEX
- EXPR – Hscript Expressions
- IMG/TIL – Image Libraries
- VEX – VEX Extensions
- SYS/UT – Basic Utilities
- Third Party Dependencies: Boost, Intel TBB, etc.



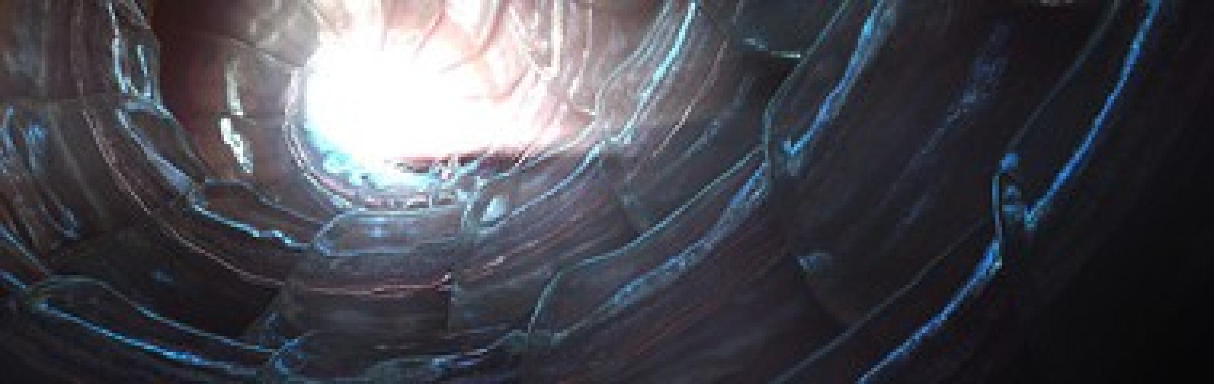
Operator Libraries

- CH – Channel (animation) Library
- PRM – Parameters
- OP – Base classes for all Houdini Nodes
  - Operator definitions
  - Node interfaces
  - Galleries - parameter presets
  - Takes - layer of parameters
- Specialized Operator Libraries
  - OBJ – Objects, SOP – Surface Ops, CHOP – Channel Ops, etc.
  - Mirror of Houdini node contexts

Node Architecture

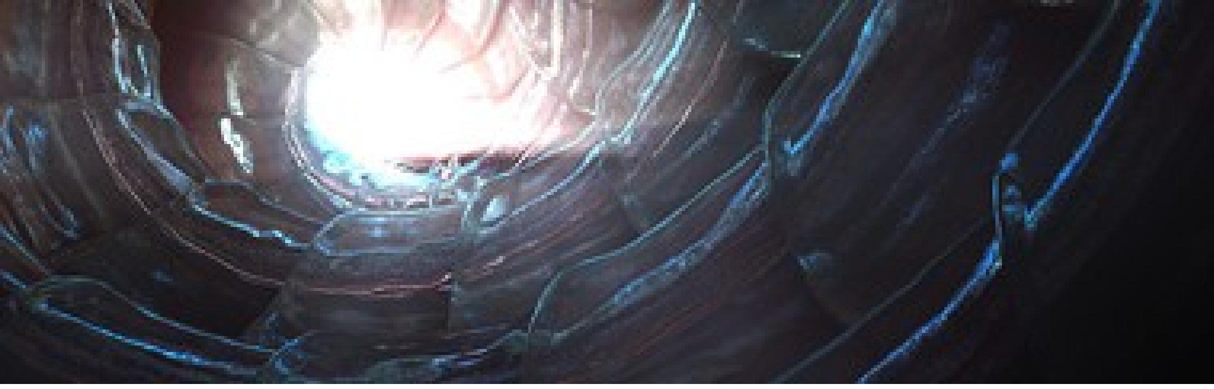


SIDE EFFECTS SOFTWARE



Minimal Anatomy of a Node

- Definition - name, number inputs
- Set of Parameters
- Method for *Cooking*
  - Cooking is dependent on what the node represents
    - OBJ: Transform
    - SOP: Geometry
    - POP: Point Geometry
    - COP: Raster Data
    - CHOP: 1D Channel Arrays



Node Registration

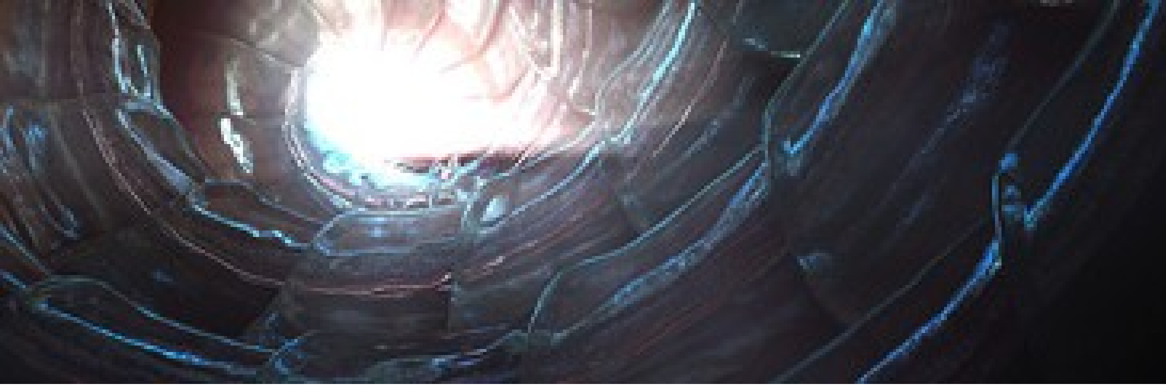
- Preset Method, Register OP_Operator with OP_OperatorTable

```
void
newSopOperator(OP_OperatorTable *table)
{
    table->addOperator(
        new OP_Operator("hdk_flatten",               // Internal Operator Type Name
                        "Flatten",                   // UI Label
                        SOP_Flatten::myConstructor,  // Class Factory
                        SOP_Flatten::myTemplateList, // Parameter Definitions
                        1,                           // Minimum # Inputs
                        1,                           // Maximum # Inputs
                        0)                           // Flags
        );
}

OP_Node *
SOP_Flatten::myConstructor(OP_Network *parent, const char *node_name, OP_Operator
*op)
{
    return new SOP_Flatten(parent, node_name, op);
}
```

Parameters Definition

- Specify Name, Type, Number of Components

```
static PRM_Name parmDist("dist",   "Distance");
static PRM_Name parmUseDir("usedir", "Use Direction Vector");

PRM_Template
SOP_Flatten::myTemplateList[] =
{
    PRM_Template(PRM_STRING,    1, &PRMgroupName, 0, &SOP_Node::pointGroupMenu),
    PRM_Template(PRM_FLT_J,     1, &parmDist, PRMzeroDefaults, 0, &PRMscaleRange),
    PRM_Template(PRM_TOGGLE,    1, &parmUseDir),
    PRM_Template(PRM_ORD,       1, &PRMorientName, 0, &PRMplaneMenu),
    PRM_Template(PRM_DIRECTION, 3, &PRMdirectionName, PRMzaxisDefaults),
    PRM_Template()              // sentinel
};
```



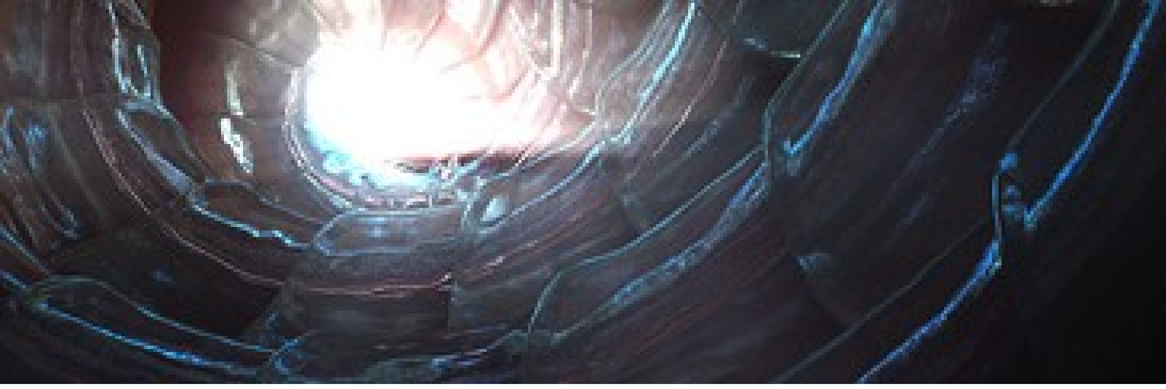
Cook Method 1

```
OP_ERROR
SOP_Flatten::cookMySop(OP_Context &context)
{
    // 1. Lock inputs, causes them to be cooked first.
    if (lockInputs(context) >= UT_ERROR_ABORT)
        return error();

    // 2. Copy input geometry into our gdp
    duplicateSource(0, context);

    // 3. Parse and create myGroup
    if (cookInputGroups(context) >= UT_ERROR_ABORT)
        return error();

    // 4. Modify gdp
    flattenGeometry();

    // 5. Unlock inputs
    unlockInputs();

    // 6. Return current error() status
    return error();
}
```

Cook Method 2

```
void
SOP_Flatten::flattenGeometry()
{
    UT_AutoInterrupt progress("Flattening Points");

    for (GA_Iterator it(gdp->getPointRange(myGroup)); !it.atEnd(); ++it)
    {
        // 1. Check if user requested abort
        if (progress.wasInterrupted())
            break;

        // 2. Get point “offset”
        GA_Offset pt_offset = *it;

        // 3. Modify position
        UT_Vector3 pos = gdp->getPos3(pt_offset);
        pos.y() = 0;
        gdp->setPos3(pt_offset, pos);
    }
}
```

Major Changes in H12 HDK



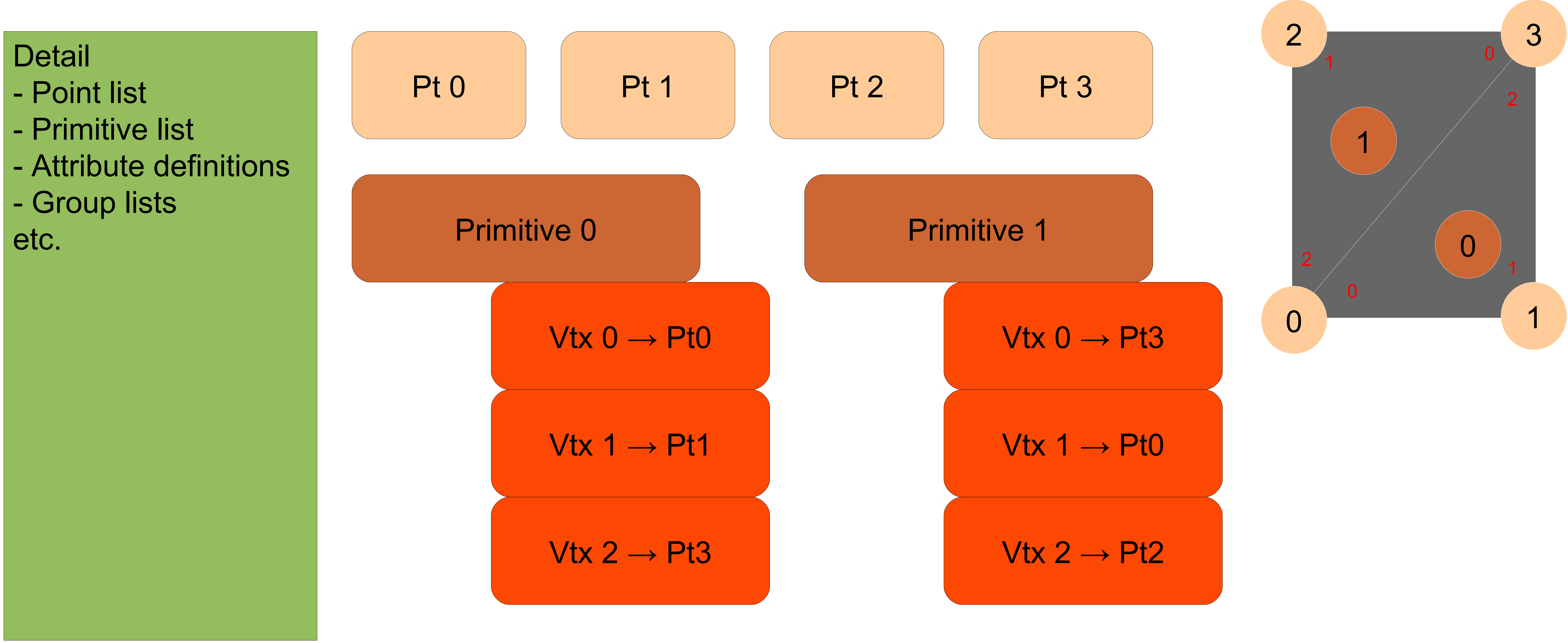
Major HDK Changes

- Rewritten Geometry Library – GA
- **`fpreal`** Numeric Type
  - H11: Single Precision
  - H12: Double Precision
- Time, Parameters, Objects, CHOPs are now fully **`fpreal`**
  - Most SOPs still use float internally
- New **`exint`** – Similar to fpreal, but for integers
- Micro-Nodes: Fine-grained dependency tracking
  - **`addParmInterest()`** replaced by new forms of **`addExtraInput()`**

New Geometry Library - GA

Geometry Structure

- GEO_Detail – Container class
- Polygon Soup - GEO_Point, GEO_Vertex, GEO_Primitive



Detail
- Point list
- Primitive list
- Attribute definitions
- Group lists
etc.

Pt 0

Pt 1

Pt 2

Pt 3

Primitive 0

Vtx 0 → Pt0

Vtx 1 → Pt1

Vtx 2 → Pt3

Primitive 1

Vtx 0 → Pt3

Vtx 1 → Pt0

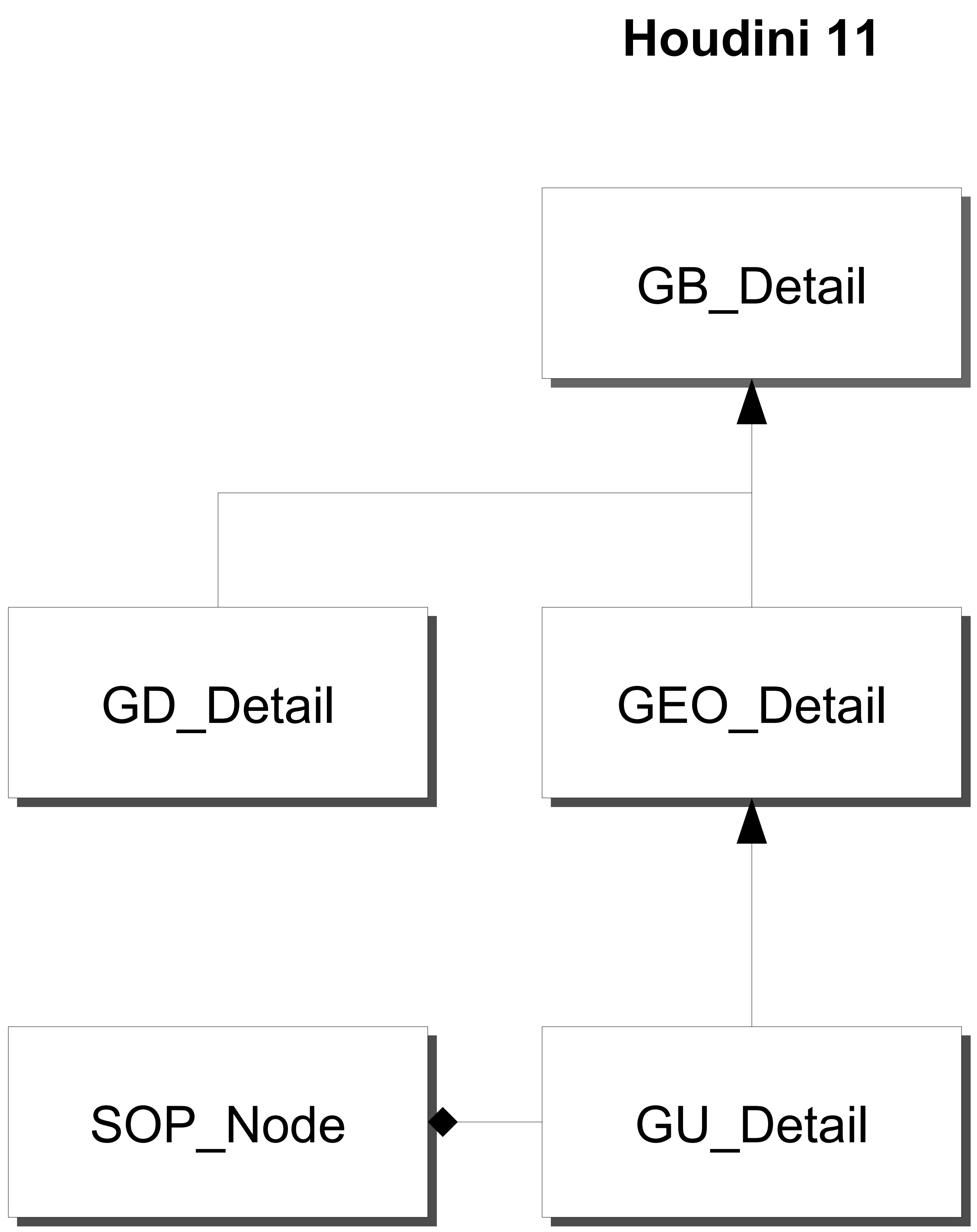
Vtx 2 → Pt2

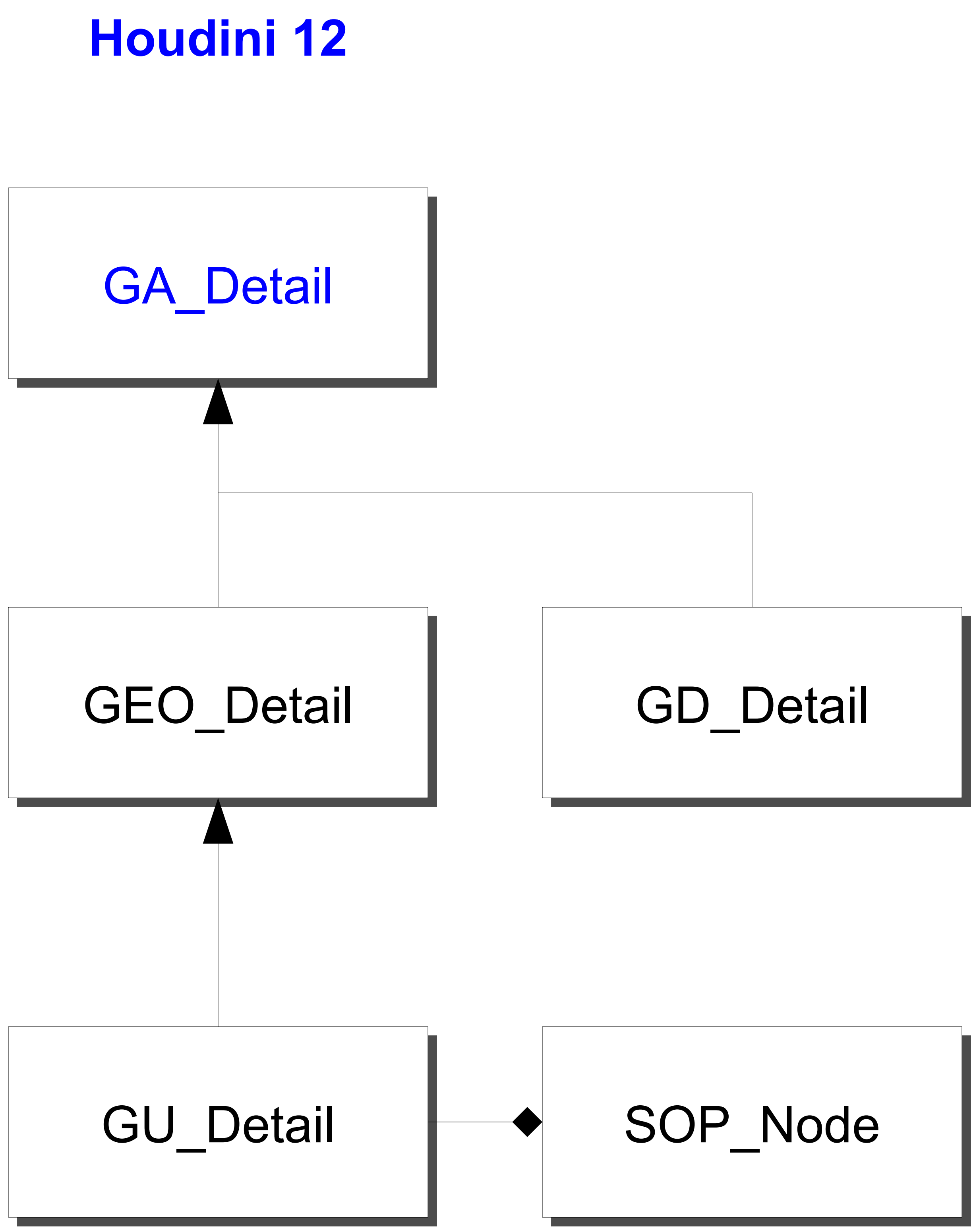
SIDE EFFECTS SOFTWARE

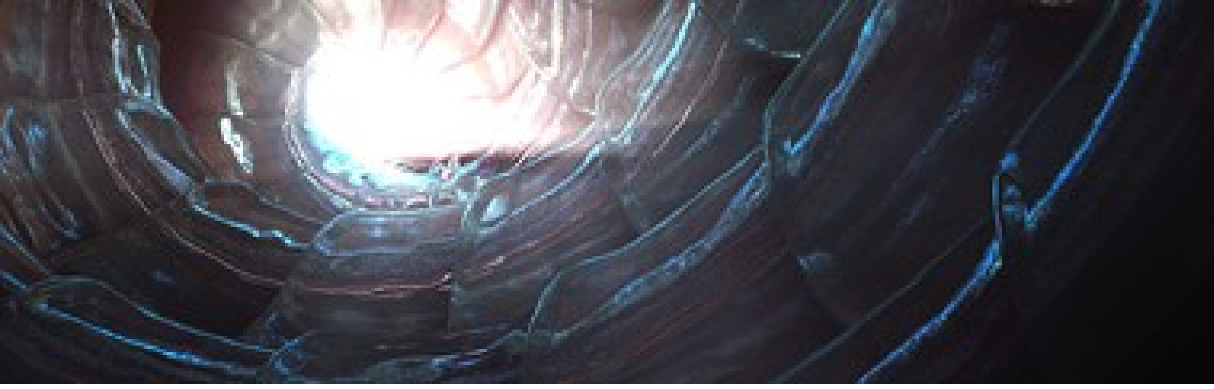
Goals of GA

- Balance two goals
  - Restructure attribute data
  - Maintain same data model (GEO_Point/GEO_Vertex). Nobody wants to port the L-System code, nor the GQ library to a different API.
- Store attributes as arrays
- GEO_Point/GEO_Vertex become “handle” objects which store offsets into the arrays
- GEO_Primitive remains largely unchanged (with possible future changes possible)

Class Hierarchy

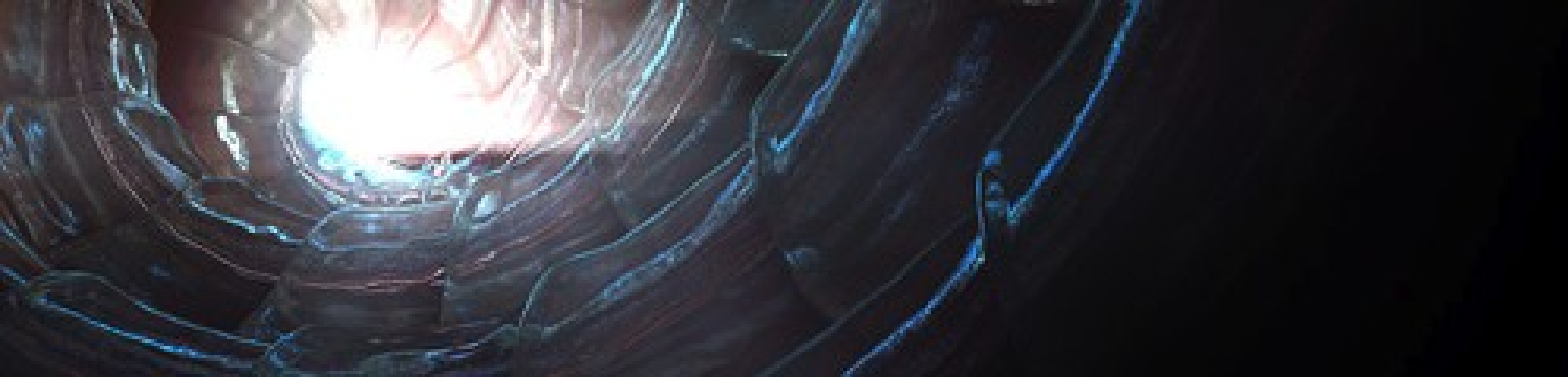






GA Changes

- P (position) is a full fledged attribute now
- Attributes not stored with elements, but in arrays
- Attributes no longer blind data (void *)
- Numeric data can have different storage (16/32/64 bit float, 8/16/32/64 bit int).
- GEO_Point and GEO_Vertex are no longer maintained by the detail.
- Vectors are vectors, normals are normals.
- No more GB_ATTRIB_MIXED
- New file format (JSON)



More GA Changes

- Point, vertex and primitive groups are now implemented with attributes
- Unique attribute names

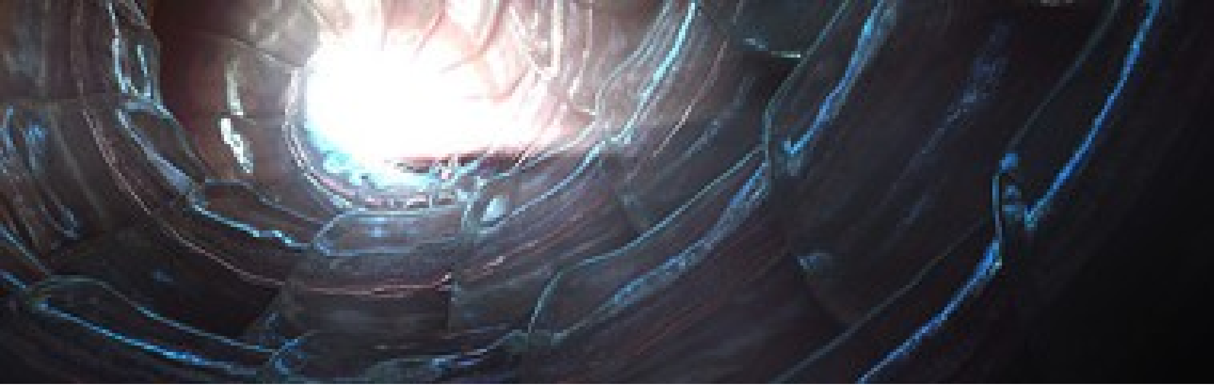
New File Format

- JSON (Javascript Object Notation)
- Binary extension (documented and Python implementation available)
- Easy to load in Python:

```
import json   # Or import hjson
geo = json.load(open(filename))
```

- Interpreting schema $HH/public/hgeo/hgeo.py

```
% hython hgeo.py
```

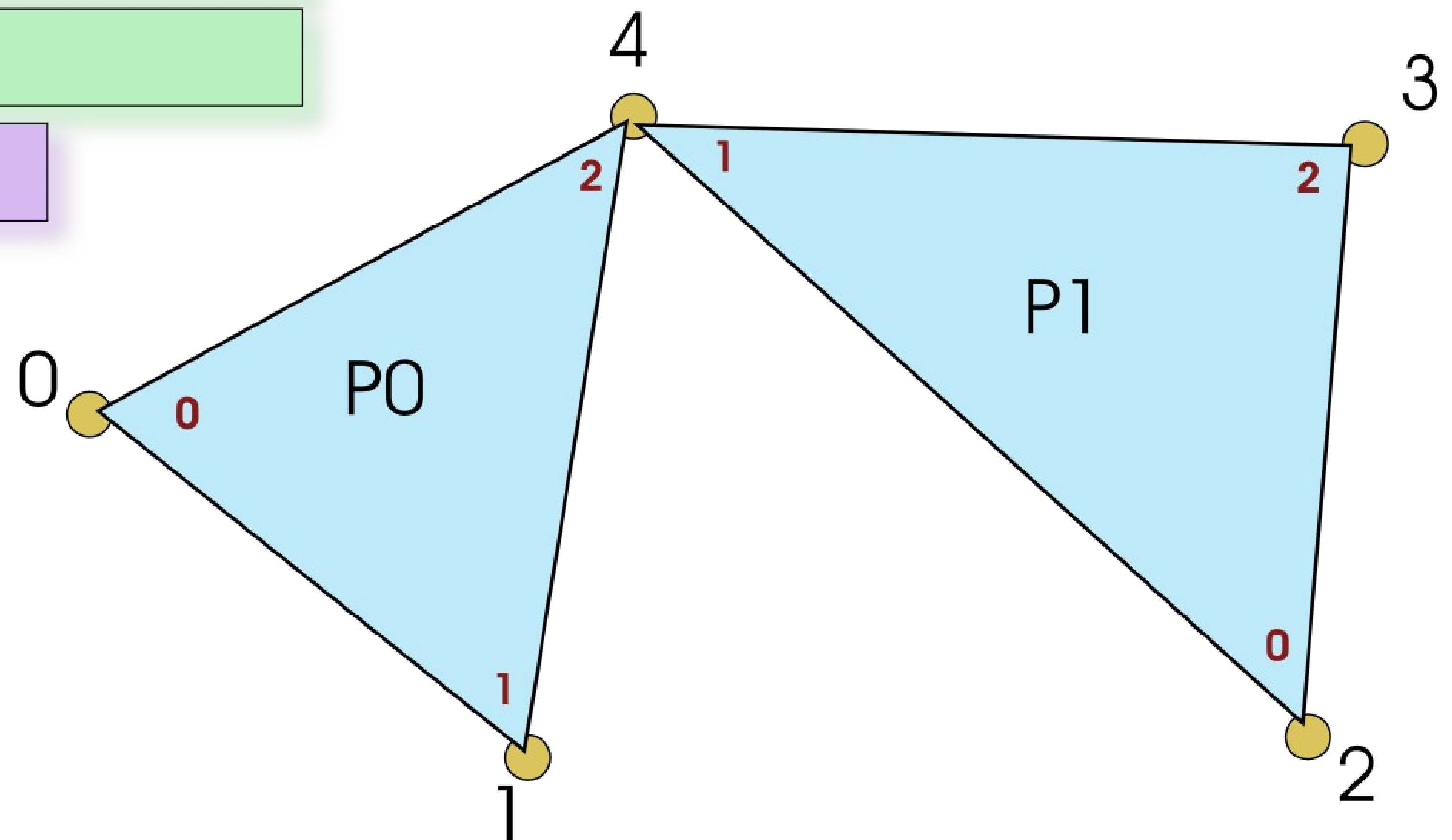


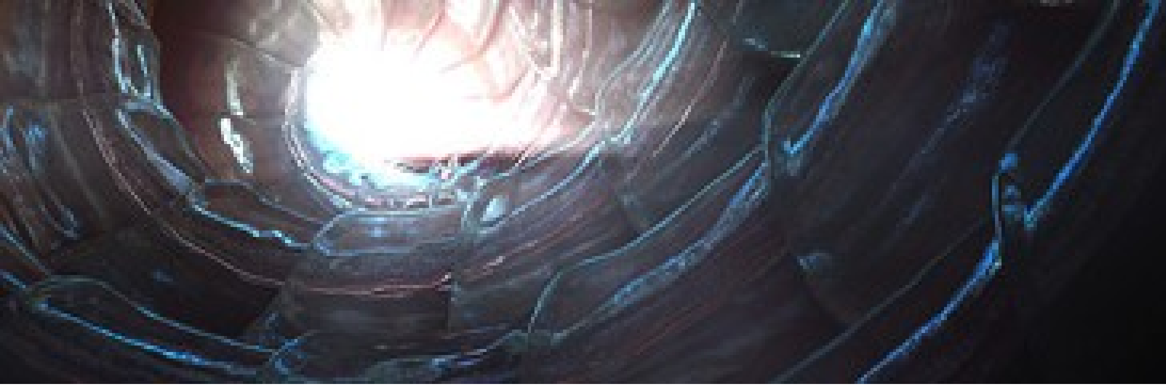
New – GA Detail

GA Detail

- Point Index Map
- Vertex Index Map
- Primitive Index Map
- Attributes
  - <P>
  - <Cd>
  - <_point_ref>
  - <uv>
  - <mass>
- Groups

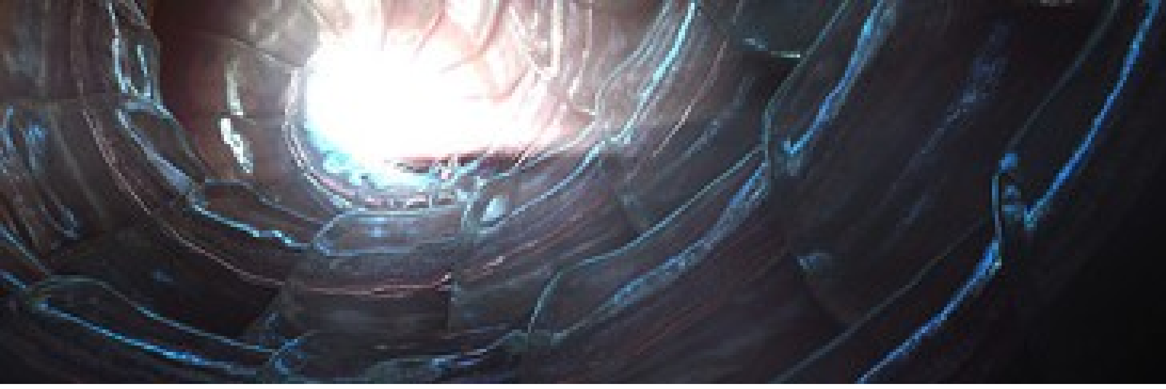
- Better attribute layout
- Less pointer chasing
- Generalized attributes (groups, topology, etc.)





GA Attributes

- Abstract attributes
- Subclasses of GA_Attribute implement specific attribute types and are called Attribute Type Implementations (ATIs)
  - Provide storage for attribute data
  - Load/Save data
  - Construct/destruct attribute data for elements
  - Provide manipulation interfaces (AIF)
- Attributes have common features
  - Name
  - Scope – private or public or group



GA Attribute Interface

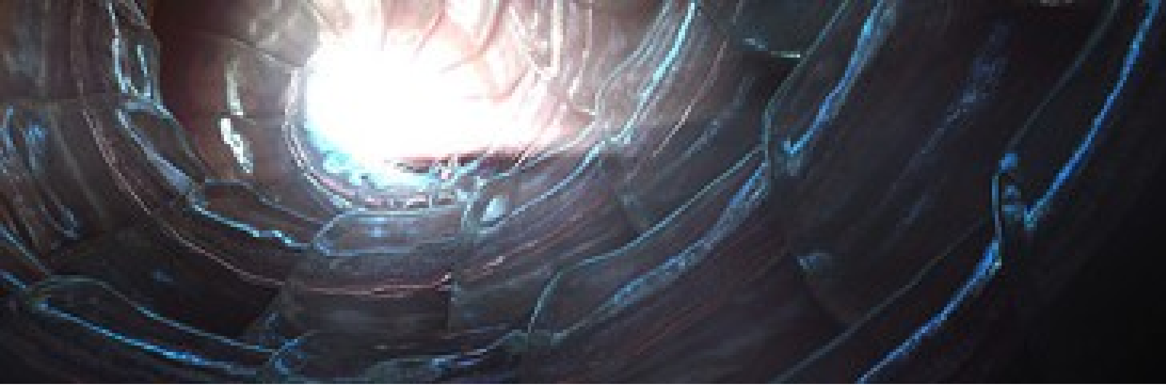
- Attributes provide implemented interfaces through Attribute Interface classes (AIF)
  - AIF_Tuple – numeric tuples
  - AIF_StringTuple/AIF_SharedStringTuple – tuples of strings
  - AIF_IndexPair – Index pair capture weights
  - AIF_Compare – Compare values
  - AIF_Merge – merge attributes
  - AIF_JSON – save/load from JSON stream
  - Etc.
- Not all AIFs need to be provided

GA Attribute Features

- GA_DataArray used by GA_ATINumeric, GA_ATIString, GA_ATIIndexPair, etc.
  - Provides different storage types: 8, 16, 32, 64 bit integers, 16, 32, 64 bit floats.
  - Paged data structure
  - COW is implemented on a per-page basis
  - Constant-page optimization (P.w, uv.z, etc.)
- GA_DataBitArray used by GA_ATIGroupBool
  - Paged bit-array
  - COW on per-page basis

Storing Your Own Data

- Alternatives to GB_ATTRIB_MIXED from H11.
- GA_ATIBlindData lets you create void data like old interface
  - GA_AIFBlindData::getReadData()
  - GA_AIFBlindData::getWriteData()
  - Provides an optional method to save/load data (unlike H11)
  - Data is opaque to geometry library, don't store pointers in data.
- GA_ATIBlob
  - Reference counted shared blobs of data for each array element
    - Blobs are destructed when no longer in use
  - GA_ATISString uses GA_ATIBlob to maintain referenced strings



Index vs Offset

- GA_IndexMap provides bidirectional look ups
  - ***GA_Index*** is the element's order
  - ***GA_Offset*** is the offset into GA_DataArray's for the element
- Access to index maps on GA_Detail

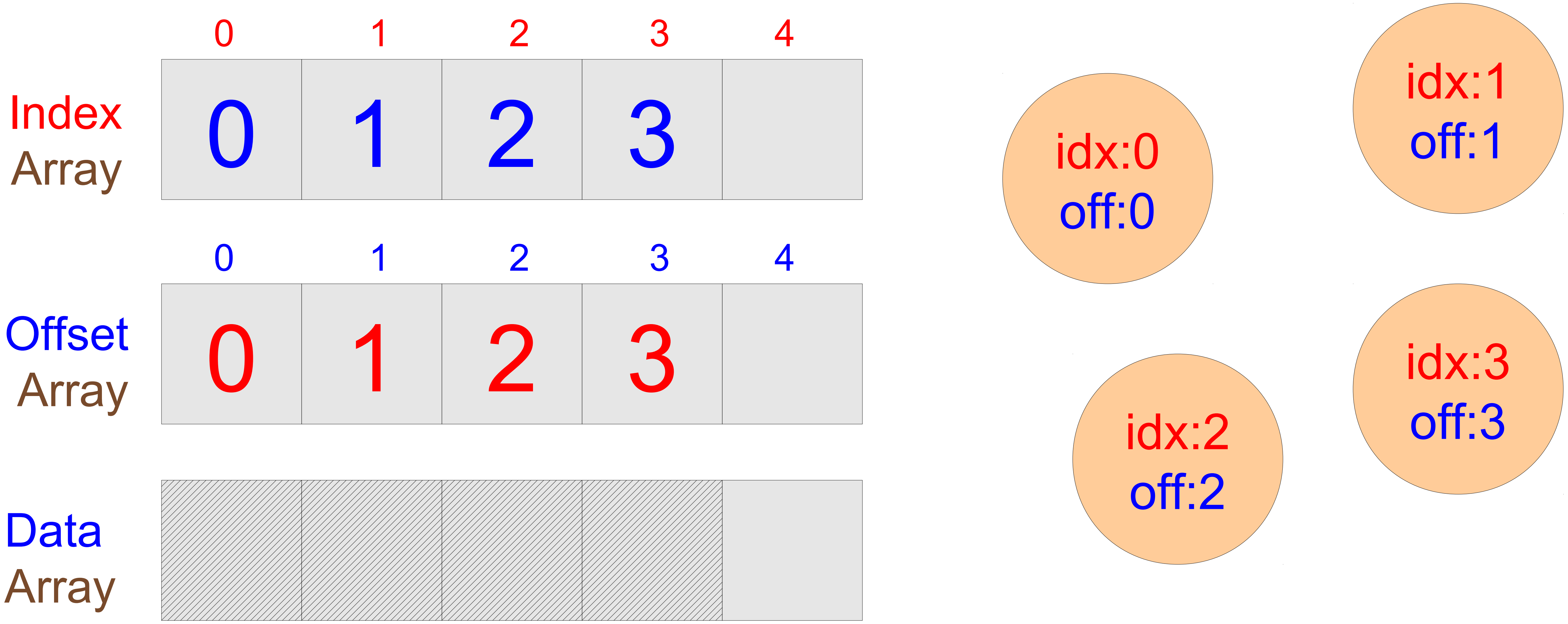
```
getPointMap(), getPrimitiveMap(), getVertexMap()
```

- Convenience Methods for conversion

```
GA_Offset offset = gdp->pointOffset(index)
GA_Index  index  = gdp->pointIndex(offset)
```

GA Index Map

- **GA_Index** is the element's order
- **GA_Offset** is the offset into attribute arrays for the element
- H11 element *indices* → H12 **GA_Index**
- H11 element *pointers* → H12 **GA_Offset**



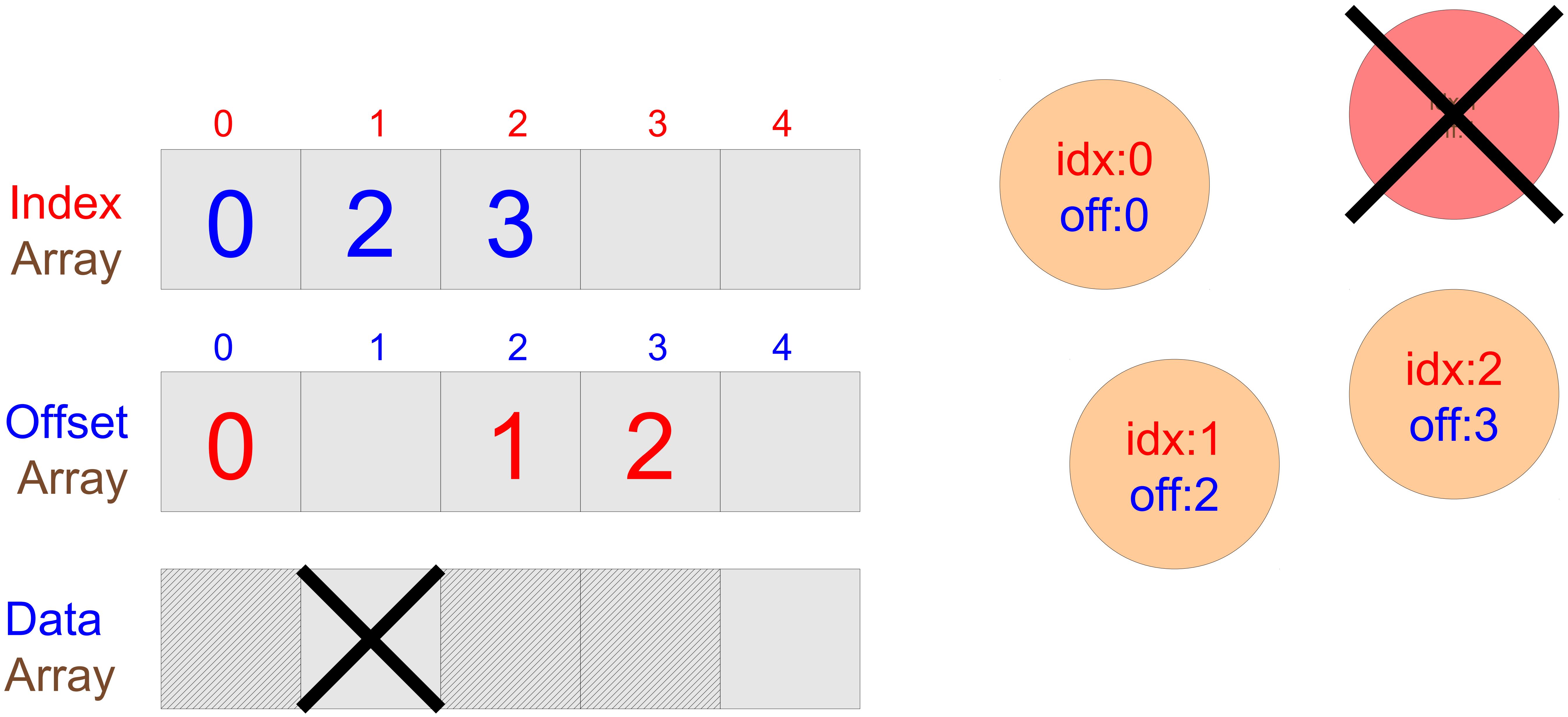
	0	1	2	3	4
Index Array	0	1	2	3	

	0	1	2	3	4
Offset Array	0	1	2	3	

Data Array	▨	▨	▨	▨	

GA Index Map

- Delete circle with GA_Index := 1
  - Element indices (GA_Index) change
  - Element offsets (GA_Offset) do **not** change



Index Array	0	1	2	3	4
	0	2	3		

Offset Array	0	1	2	3	4
	0		1	2	

Data Array					
	▨	☒	▨	▨	

Fragmentation

- Deleting elements leaves holes in the offset and data arrays
- Holes are **not** reused
  - maintains offset monotonicity
- Explicit de-fragmentation
  - May occur automatically after SOP cook
  - May harden shared pages

Accessing Attribute Data

Access Attribute Data: GA_GBElement

- Familiar interface for porting from GB code

```
GB_AttributeRef h;

h = gdp->findPointAttrib(
             “Cd”,
             sizeof(float)*3,
             GB_ATTRIB_FLOAT);
if (h.isValid())
{
    FOR_ALL_GPOINTS(gdp, ppt)
    {
        ppt->setValue<UT_Vector3>(
            h, UT_Vector3(1,0,0));
    }
}
```

```
GA_RWAttributeRef h;

h = gdp->findFloatTuple(
             GA_ATTRIB_POINT,
             “Cd”,
             3);
if (h.isValid())
{
    FOR_ALL_GPOINTS(gdp, ppt)
    {
        ppt->setValue<UT_Vector3>(
            h, UT_Vector3(1,0,0));
    }
}
```

GA_GBElement Methods

- GA_GBElement::getValue<POD>(h)
  - GA_ROAttributeRef, GA_RWAttributeRef
- GA_GBElement::setValue<POD>(h, const &v)
  - GA_WOAttributeRef, GA_RWAttributeRef
- Use AIF interface for implementation
- Generic (works with all attributes) but virtual thunks

AIF interface using GA_Offset (not GA_GBElement)

- Direct interface to AIF's (similar to GA_GBElement in performance)
  - Deals with tuples of POD (no complex types)
  - Each tuple call is a virtual thunk

```
GA_RWAttributeRef  h = gdp->findFloatTuple(GA_ATTRIB_POINT, “N”, 3);
GA_Attribute      *attrib = h.getAttribute(); // This is the ATI
const GA_AIFTuple *tuple = attrib->getAIFTuple();

for (GA_Iterator it(gdp->getPointRange()); !it.atEnd(); ++it)
{
    GA_Offsets     offset = *it;     // Get array offset for point
    UT_Vector3     N;
    tuple->get(attrib, offset, N.data(), 3);
    N.normalize();
    tuple->set(attrib, offset, N.data(), 3);
}
```

SIDE EFFECTS
SOFTWARE

Faster Access to Data

- GEO_Point/GEO_Vertex (GA_GBElement) provides easy interface for porting
- Not most efficient interface – code will be slower than H11 (though less memory)
- More direct access to attribute data is needed

GA_Handle

- GA_Handle for GA_ATINumeric data only
- Not only is the code simpler, it's faster too!
  - Caveat: GA_ATINumeric data only (though GA_*HandleS exists for strings)

```
GA_RWAttributeRef  h = gdp->findFloatTuple(GA_ATTRIB_POINT, “N”, 3);
GA_RWHandleV3      N_h(h.getAttribute());

for (GA_Iterator it(gdp->getPointRange()); !it.atEnd(); ++it)
{
    UT_Vector3 N = N_h.get(*it);
    N.normalize();
    N_h.set(*it, N);
}
```

Iterator Cost

- Recall that GA_DataArray is paged and may have “holes”
- For various reasons (see future slide), you may want to iterate over contiguous “pages” of GA_Offsets.
- The fastest access to data arrays uses GA_PageHandle.

  User experience shows that switching between GA_Handle to GA_PageHandle may provide improvement of up to 5%. It's up to you whether you feel the code complexity is worth the extra effort.

GA_PageHandle

- GA_PageHandle for GA_ATINumeric data only
  - Minimal iterator cost
  - Direct access to data

```
GA_RWAttributeRef h = gdp->findFloatTuple(GA_ATTRIB_POINT, “N”, 3);
GA_RWPageHandleV3 N_ph(h.getAttribute());
GA_Offset         start, end;

for (GA_Iterator it(gdp->getPointRange()); it.blockAdvance(start, end); )
{
    N_ph.setPage(start);
    for (GA_Offset ptoff = start; ptoff < end; ++ptoff)
        N_ph.value(ptoff).normalize();
}

// inline T &value(GA_Offset off) { return myPagePtr[off-myBaseOffset]; }
```

Porting Guide Online

- http://www.sidefx.com/docs/hdk12.0/

  (needs beta access on forum)
- HDK > Houdini Geometry > GA Porting Guide
  - Provides a porting cook-book. Common patterns we found when doing our port from H11 → H12.
- Example - destroying unused points:

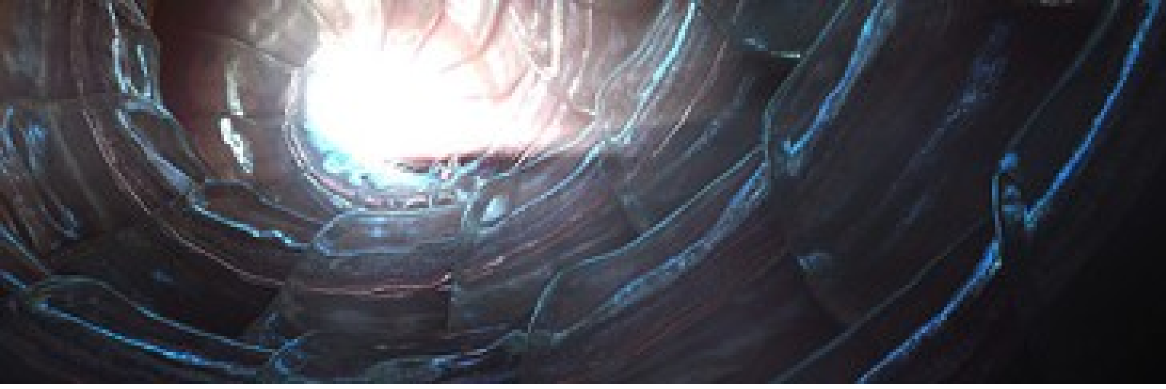
```
// GB Code
gdp.removeUnusedPoints(group);
==>
// GA Code
gdp.destroyUnusedPoints(group);
```

Optimization

Iterators

- In H11, iteration was done with macros or explicitly

```
// Iterate by macro
FOR_ALL_GPOINTS(gdp, ppt)
{
    ...
}

// Iterate by point number/index
for (int i = 0; i < gdp->points().entries(); ++i)
{
    ppt = gdp->points()(i);
    ...
}

// Group traversal iteration
for (int i = 0; i < gdp->points().entries(); ++i)
{
    ppt = gdp->points()(i);
    if (!myGroup->contains(*ppt))
        continue;
    ...
}
```



Iterators

- Use GA_Iterator in new code. Optimized group traversal.

```
// group can be NULL
for (GA_Iterator it(gdp->getPointRange(group)); !it.atEnd(); ++it)
{
    GA_Offset  ptoff = *it;
    ...
}

// Use GA_PageHandle for fastest numeric attribute access
for (GA_Iterator it(gdp->getPointRange(group)); it.blockAdvance(start, end);)
{
    page_handle.setPage(start);

    const fpreal32 *data_ptr = &page_handle.value(start);
    GA_Size         n = end - start;
    ...
}
```

GEO_Point

- GEO_Vertex/GEO_Point are now lightweight handles.
  - Can be created as stack objects
- Existing code passes GEO_Point * objects. Some lower level code holds onto these references.
- **`gdp->points()`** allocates points on demand and holds them
  - 16 bytes per point
- There may be locks for thread safe access of point data
- Quick fix is possible for code using macros...

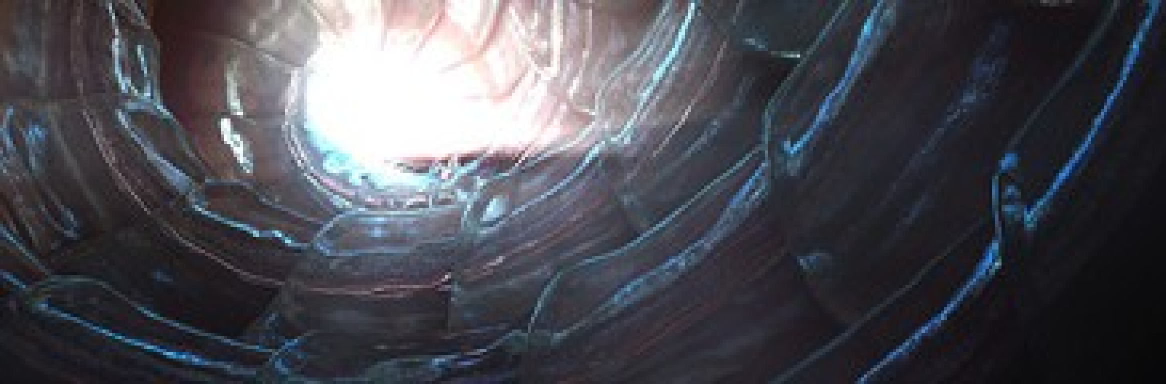
Simple Optimization For GA_GBElement

- Provided nothing in the loop holds onto a reference...

```
GA_RWAttributeRef h;
h = gdp->findFloatAttribute(
             GA_ATTRIB_POINT,
             “Cd”,
             3);
if (h.isValid())
{
    GEO_Point *ppt;
    GA_FOR_ALL_GPOINTS(
        gdp, ppt)
    {
        ppt->setValue<UT_Vector3>(h,
                UT_Vector3(1,0,0);
    }
}
```

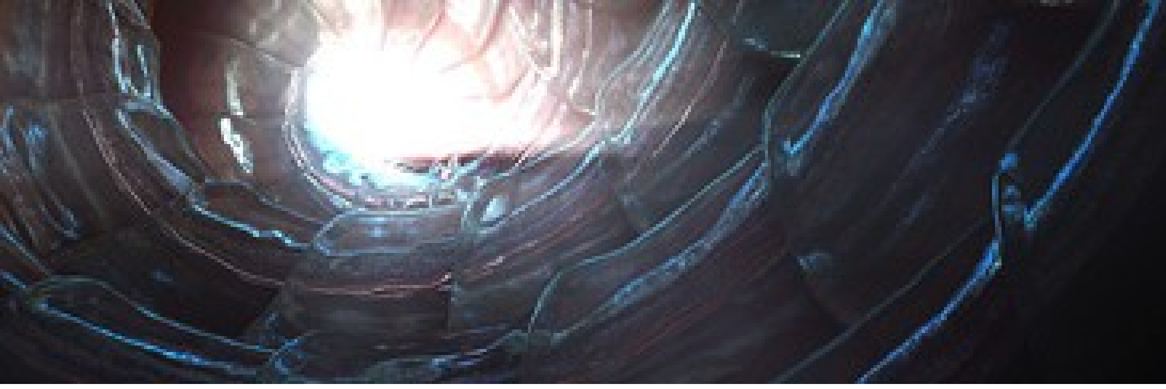
```
GA_RWAttributeRef h;
h = gdp->findFloatTuple(
             GA_ATTRIB_POINT,
             “Cd”,
             3);
if (h.isValid())
{
    GA_FOR_ALL_GPOINTS_NC(
        gdp, GEO_Point, ppt)
    {
        ppt->setValue<UT_Vector3>(h,
                UT_Vector3(1,0,0);
    }
}
```

SIDE EFFECTS
SOFTWARE



GEO_PointList

- In GB, the GEO_PointList was an array of pointers holding references to GEO_Point
- In GA, GEO_PointList is a “handle” to the GA_IndexMap
- Old code might have used GEO_PointList as a container
  - No longer a container (“handle”)
  - No elements to store any more
  - Consider GA_OffsetArray
- That is code that has `gdp.points()` is creating a temporary handle.



Vertex Iteration

- GB

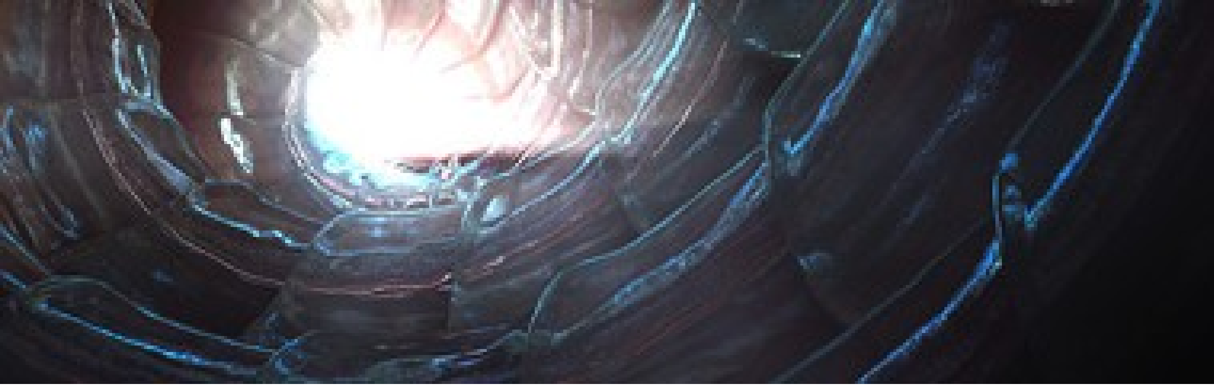
```
for (int i = 0; i < prim->getVertexCount(); i++)
    doSomething(prim->getVertex(i));
```

- GA

```
for (int i = 0; i < prim->getVertexCount(); i++)
    doSomething(prim->getVertexElement(i));
```

```
for (GA_Iterator it(prim->getVertexRange()); !it.atEnd; ++it)
    doSomething(it.getOffset());
```

```
GA_Primitive::const_iterator    it;
for (prim->beginVertex(it); !it.atEnd; ++it)
    doSomething(it.getVertexOffset());
```



Copy

- GA_GBPoint::copyPoint(), GA_GBVertex::copyVertex(), GEO_Detail::copyPointAttributes(), GEO_Detail::copyVertexAttributes(), GEO_Detail::copyPrimitiveAttributes(), etc
  - Called per-element
    - Filtering overhead
    - Additional look-up overhead
- GA_AttributeRefMap
  - Filtering and look-up done once
  - Special-cased to avoid virtual calls for standard numeric attributes

Wranglers

- Designed to ease sharing of GA_AttributeRefMap objects across multiple functions
- Instantiate cache at a high level

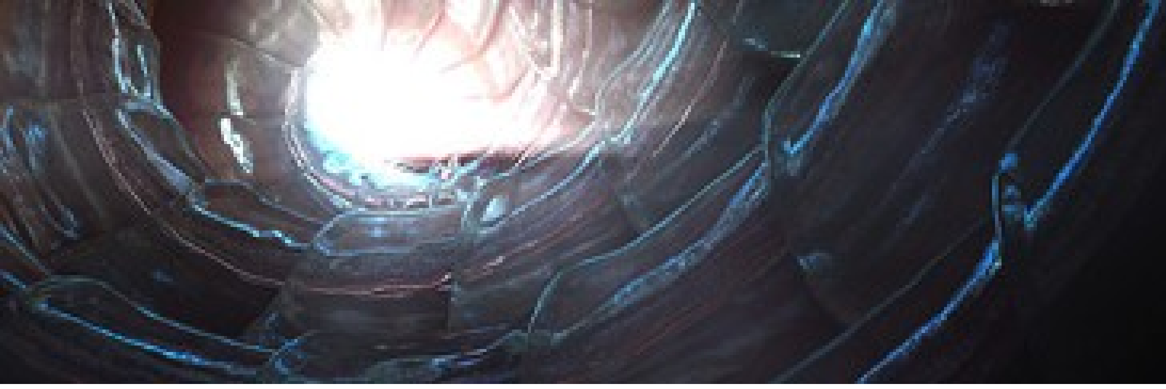
```
GA_ElementWranglerCache wranglers(gdp, GA_PointWrangler::INCLUDE_P);
```

- Pass down and use at lower levels

```
wranglers.getVertex().copyAttributeValues(dest_vertex, src_vertex);
wranglers.getPoint().copyAttributeValues(dest_point, src_point);
```

Threading

- Paged data structures
  - Write access split over pages – i.e. only one thread can write to a page at one time (constant page expansion etc.)
  - When an attribute is read/write, only one thread can read/write to a page at one time
  - When a page is read-only, any number of threads can operate on the attribute
- Two threading models supported



TBB Threading

- Intel's TBB library wrapped in UTparallelFor() and UTparallelReduce()
- GA_SplittableRange is a GA_Range which conforms to TBB splittable concept and can always be split()

```
Class Task {
    void operator()(const GA_SplittableRange &r) const
    {
        for (GA_Iterator it(r); !it.atEnd(); ++it)
        {
            …
        }
    }
}

UTparallelFor(GA_SplittableRange(range), Task());
```

UT Threaded Algorithm

- Using UT_ThreadedAlgorithm

```
Class TaskAlgorithm
{
    THREADED_ALGORITHM2(
          task, range.canMultiThread(),
          const GA_SplittableRange &, range,
          const TaskParms &, parms)

    void taskPartial(
          const GA_SplittableRange &range,
          const TaskParms &tdata, const,
          UT_JobInfo &info)
     {
         for (GA_PageIterator pit(range.beginPages(info)); !pit.atEnd(); ++pit)
         {
             for (GA_Iterator it(pit); !it.atEnd(); ++it)
             { … }
         }
     }
}
```