We are pleased to share our latest developments in medical computing, HPC and visualization, software process, data and analytics, and computer vision, which include releases for ITK, CMake, and Midas Platform. We are also pleased to announce our participation in several projects that aim to address pressing needs in various communities in academia and industry.

To remain up-to-date on our advancements in open-source software and state-of-the-art technology, be sure to catch us at the events we attend this summer and fall. A list of upcoming events can be found on www.kitware.com/events. If you would like to set up a time to meet with us at any of the listed events to discuss employment opportunities, potential collaboration, or consulting services, please contact kitware@kitware.com.

Want to help us develop cutting-edge solutions and work on research problems alongside leading developers and computational scientists? We are currently hiring for several positions. Please visit our employment site for details.

ITK 4.8 RELEASED
On behalf of the Insight Toolkit community, we are pleased to announce the release of ITK 4.8.0. This release features a Python-wrapping infrastructure based on CastXML, which works with the latest Microsoft Visual C++ (MSVC), Clang, and GNU Compiler Collection (GCC) compilers. The release also brings new remote modules such as BridgeNumPy, LabelErodeDilate, and ParabolicMorphology; and it enables more modules to be built as shared libraries.

ITK 4.8 includes enhanced point set registration capabilities, along with experimental cross-compilation support for Windows (MinGW-w64), ARMv6 (Raspberry Pi), ARMv7 (Android), ppc64le (POWER8), and JavaScript (Emscripten). The release also includes itk::FFTPadImageFilter, which allows for automatically padding images for the greatest prime factor supported by the fast Fourier transform (FFT) implementation.

Congratulations and thank you to those who contributed to version 4.8. We would especially like to recognize new ITK contributors: David Froger, Cyril Mory, Dzenan Zukic, Ivan Setiawan, Jan Bergmeier, Rolf Eike Beer, Davis Vigneault, Gary Jia, and Alexander Hewer.

To download the latest release, please go to http://itk.org/ITK/resources/software.html.
CMAKE 3.3 RELEASED

CMake 3.3 is now available. The following are notable changes for the release.

The if() command knows a new IN_LIST operator that evaluates to true if a given element is contained in a named list, and the add_dependencies() command knows to allow dependencies to be added to interface libraries. In addition, the find_library(), find_path(), and find_file() commands now search in installation prefixes derived from the PATH environment variable.

With the 3.3. release, the <LANG>_VISIBILITY_PRESET and VISIBILITY_INLINES_HIDDEN target properties affect compilation in sources of all target types; while a <LANG>_INCLUDE_WHAT_YOU_USE target property and supporting CMAKE_<LANG>_INCLUDE_WHAT_YOU_USE variable tell the Makefile and Ninja generators to run include-what-you-use, along with the compiler for C and CXX languages.

Depreciated features in version 3.3 include Visual Studio 6 and Visual Studio 7 generators. Furthermore, the commands ctest_build() and build_command() no longer tell "make" tools to ignore errors with the –i option.

For full release notes, sources, binaries, and documentation, please visit http://www.cmake.org.

GIRDER 1.3 RELEASED

Kitware is pleased to announce the release of Girder 1.3, a data science tool that supports scalable data management solutions. With this release, Girder can run on both Python 2.7 and Python 3.4. Also new with this release is the ability of clients to upload to an S3 asset store, as they would any other type of asset store, if they do not implement the direct-to-S3 upload behavior.

Further highlights of the release include the ability via the Representational State Transfer (REST) application programming interface (API) and the web client to apply an access control list to all children of a collection or folder recursively, the ability via the REST API and the web client to download an entire collection as a zip archive, and the addition of utilities to the JavaScript library for dynamically changing the page layout during navigation.

For more information on Girder 1.3, and to download the latest release, please visit http://www.tangelohub.org/girder.

MIDAS SERVER 3.4 RELEASED

Midas Server 3.4 provides support for SQLite databases and for the Google Cloud Platform, including App Engine and Cloud Storage. It allows for styling using Bourbon and Sass, as well as provisioning using Ansible and Vagrant. In addition, the release features bug fixes; dependency management using Composer and Packagist; and improvements to e-mail notifications, thumbnail creation, performance, and security.

The Midas Platform is Kitware’s versatile, open-source, web-enabled solution that provides a cohesive system for managing, visualizing, and processing data. For more information on the Midas Platform, and to download the 3.4 release of Midas Server, please visit http://www.kitware.com/opensource/midasplatform.html.

UV-CDAT 2.2 RELEASED

Ultrascale Visualization Climate Data Analysis Tools (UV-CDAT) is a system designed to aid climate scientists in handling big data analytics and sensitivity analyses using heterogeneous data sources from multiple disciplinary domains. A major highlight of the 2.2 release is support for generating one- and two-dimensional plots, which was added by implementing VTK in the system’s backend.

What is more, an Askbot website (http://askbot-uv-cdat.llnl.gov/) is now available to members of the UV-CDAT community. While the website currently supports UV-CDAT 2.2, it will also support future releases.

For more information on UV-CDAT and features of the 2.2 release, please visit https://github.com/UV-CDAT.

RGG 2.0 RELEASED

Reactor Geometry Generator (RGG) 2.0 includes boundary layer support and a new file format. With this new file format, the user need only maintain a single reactor model file. All of the necessary input files required by MeshKit’s AssyGen and CoreGen tools are now automatically generated. In addition, the new file format allows for sharing reactors, pins, and ducts among various subassemblies; and it provides for creating and sharing libraries of pins, ducts, and materials among different core models.

In RGG 2.0, sectioning is automatically completed for the user, both visually and in the creation of input files related to MeshKit. Version 2.0 also offers the ability to view models and resulting meshes; the ability to load in meshes created outside of RGG; the ability to view meshes based on material, volume, and side-set classifications; and the ability to save meshes and subsets in Mesh-Oriented datABase (MOAB) and Exodus formats.

To download the latest RGG release, please visit http://www.computationalmodelbuilder.org/rgg.
INTRODUCTION

The Visualization Toolkit (VTK) system has been in active development for over 20 years, and from the earliest days, the community has concerned itself with parallel computing. The class vtkMultiThreader was introduced in 1997, mainly to support threaded, shared-memory data processing in the imaging pipeline. Scalable, distributed parallel computing has also been a major focus. Applications such as ParaView depend on these capabilities to execute on large leadership computing facilities. For example, ParaView/VTK has been successfully run using 256K processors in a coprocessing simulation with the Phasta turbulent flow analysis code on Mira, Argonne National Laboratory’s IBM Blue Gene/Q system [1]. Such data-parallel, distributed computing applications require a sophisticated execution model, and VTK has seen several incarnations of the pipeline architecture as a result.

Similar development efforts continue to this day, with client-server models extending the reach of server-side parallel computing through systems such as VTK/ParaViewWeb [2]. In fact, with the advent of cloud computing and systems such as Amazon Web Services (AWS) and Google Compute Engine, it is now possible for many organizations to use VTK/ParaView in a distributed, customized, relatively inexpensive, on-demand basis to meet computational needs.

However, as we all know, the computing industry is not standing still. The field of parallel computing is advancing rapidly due to innovations in graphics processor unit (GPU) and multi-core technologies. With the plateauing of serial clock speeds, due to power and other physical constraints, chip manufacturers are adding computing capacity in the form of additional computing cores and vectorized processing units. While such hardware represents a significant leap in computing potential—with the distinct possibility of a dawning exascale computing era—these hardware advances pose significant challenges to software systems. Not only are popular programming languages unsuited for most parallel computing applications, but the diversity of emerging parallel approaches makes it very difficult to create general programs that can take advantage of different hardware and software libraries across a variety of platforms.

There is also the algorithm design challenge: Many developers simply adapt applications that were initially developed in a serial computing environment. Such adaptations typically use relatively limited task- and data-parallel approaches that require significant synchronization and inter-process communication, as well as locks and mutexes to access serial data structures. These approaches often do not scale well on massively-parallel hardware. Instead, in the interest of scaling across the emerging parallel hardware, developers must rethink parallel algorithms that may be slower and less efficient at a small scale, but that execute efficiently as data becomes large.

These challenges are significant and require a concerted effort to realize the full potential of parallel computing. The VTK/ParaView community—led by Dr. Berk Geveci at Kitware, in collaboration with prestigious computing organizations such as Department of Energy National Laboratories—is taking these challenges to heart and has developed a broad vision for the future. This vision is one that depends on a multi-pronged approach to address differing parallel computing needs. The major components of this vision include the following:

1. vtkSMPTools
   This is an abstraction for threaded processing, which under the hood uses different libraries such as Threaded Building Blocks (TBB), OpenMP, and X-Kaapi. The typical target application is coarse-grained shared-memory computing as provided by mainstream multi-core, threaded central processing units (CPUs) such as Intel’s i5 and i7 architectures.

2. VTK-m
   This is a toolkit of scientific visualization algorithms for emerging processor architectures (GPUs and coprocessors). VTK-m is designed for fine-grained concurrency and provides abstract data and execution models that can be utilized to construct a variety of scalable algorithms.

3. New Algorithms
   This component requires the redesign and implementation of algorithms and data structures that best take advantage of new parallel computing hardware and libraries. To do so, it is necessary to identify essential capabilities for a variety of visualization and data analysis tasks.

Currently, all of these components are under active development. This article addresses vtkSMPTools due to its relative simplicity and maturity. Future articles will describe the other components (VTK-m and new algorithms) as they mature and are further incorporated into VTK and ParaView.

VTKSMPTools

The best place to start learning about vtkSMPTools is a short but informative description, titled "Parallel Processing with the SMP Framework in VTK" [3]. As stated in the introduction to this document, the objective of the symmetric
multiprocessing (SMP) framework is to provide a basic infrastructure for the easy development of shared-memory parallel algorithms in VTK. The resulting framework is quite simple, supporting the following:

- atomic integers and associated operations
- thread local storage
- parallel building blocks such as parallel for loops

A noteworthy feature of vtkSMPTools is that it defines an abstraction to a variety of underlying threading systems. At compile time (using the CMake advanced option VTK_SMP_IMPLEMENTATION_TYPE), it is possible to choose one of several possible threading facilities. Currently, the choices are the Intel TBB library [4], Kaapi [5], Simple (soon to be replaced by the OpenMP backend), and Sequential (as in serial processing). As noted, OpenMP is being added to the mix and will be available soon. In addition, key general-purpose algorithms such as parallel_sort will be supported by vtkSMPTools.

With vtkSMPTools, rather than worrying about scheduling and balancing work among multiple threads, a developer can focus on writing algorithms as a sequence of small, independent tasks. vtkSMPTools takes care of distributing the work to the underlying thread resources.

**PARALLEL FOR()**

The vtkSMPTools framework implements a thin wrapper around existing tools such as TBB and OpenMP, and it provides a simple application programming interface (API) for writing parallel code. Useful algorithms that scale well can be created using the parallel for loop

```
vtkSMPTools::For(begin,end,func);
```

The arguments to For() consist of start and end parameters, which are expressed as vtkIdType integers and define the semi-open range [start,end), and a functor to be executed over this range. The functor is a C++ class with an implementation of operator(). Optionally, a “chunk” or “grain” size can be provided that offers a hint as to how to divide up the problem for effective load balancing versus scheduling overhead.

In practice, the parallel for construct looks like this:

```
vtkSomeFunctor func;
vtkSMPTools::For(begin,end,func);
```

This high-level specification of the computation enables the different backends to employ various scheduling strategies for optimal performance. For example, the OpenMP backend supports static, dynamic, guided, or implementation-dependent default scheduling strategies, which can be selected at runtime by setting the OMP_SCHEDULE environment variable. (Please refer to the OpenMP specification for a description of these strategies.) The TBB backend implements a technique called work stealing [6]. In work stealing, each thread has its own work pool, from which it takes the next task to execute and adds any new tasks that are generated. Using local queues removes contention, which is present when using a single work pool. When a thread has exhausted its own pool, it steals more work from other threads’ pools, and hence, balances the workload.

The optional grain size parameter to the parallel for construct provides a hint regarding the granularity of the computation to be performed. This is useful for equilibrium between scheduling overhead and load balancing. Parallelism is achieved by splitting the input range into smaller sub-ranges based on the grain size and assigning these sub-ranges to the threads. If the grain size is too low, the functor is invoked many times, and the overhead of invocation adds up, resulting in poor performance. If the grain size is too high, the overhead is reduced. This can result in poor load balancing, as some threads may finish their work sooner (if particular sub-ranges are less demanding) and become idle, since there are not enough sub-ranges to keep them fed.

The grain size depends on the type of computation, and it is recommended that an appropriate value be specified. The TBB backend can chose a grain size heuristically (and adaptively adjust it), but this feature is not available on other backends discussed in this article. Other backends simply assign tasks based on a small multiple of the number of available threads.

**EXAMPLE**

The following is a simple example, inspired by the VTK class vtkWarpVector, for creating the class vtkSMPWarpVector.

```
#include "vtkSMPTools.h"

template <class T1, class T2>
class vtkSMPWarpVectorOp
{
  public:
    T1 *InPoints;
    T1 *OutPoints;
    T2 *InVector;
    T1 scaleFactor;
  
  void operator()(vtkIdType begin, vtkIdType end)
  {
    T1* inPts = this->InPoints + 3*begin;
    T1* outPts = this->OutPoints + 3*begin;
    T2* inVec = this->InVector + 3*begin;
    T1 sf = this->scaleFactor;
    vtkIdType size = 3*(end-begin);
    vtkIdType nend = begin + size;

    for (vtkIdType i = begin; i < nend; i++)
    {
      T1 scaleFactor = sf / 3;
      T2* inVec = this->InVector + 3*i;
      T1* inPts = this->InPoints + 3*i;
      T1* outPts = this->OutPoints + 3*i;
      T1* inVec = this->InVector + 3*i;
      this->scaleFactor = scaleFactor;
      this->warpVector(inVec, inPts, outPts);
    }
  }
};
```
for (vtkIdType index = begin; index < nend; index++)
{
    *outPts = *inPts + sf * (T1)(*inVec);
    inPts++; outPts++; inVec++;
}
}

Next, in the VTK class, the functor is used as follows (in the templated execution method `vtkSMPWarpVectorExecute2`):

```cpp
template <class T1, class T2>
void vtkSMPWarpVectorExecute2(vtkSMPWarpVector *self, T1 *inIter, T1 *outIter,
T2 *inVecIter, vtkIdType size)
{
    vtkSMPWarpVectorOp<T1, T2> func;
    func.InPoints = inIter;
    func.OutPoints = outIter;
    func.InVector = inVecIter;
    func.scaleFactor = (T1)self->GetScaleFactor();

    vtkSMPTools::For(0, size, func);
}
```

(The full implementation is available in `Filters/SMP/vtkSMPWarpVector.h/.cxx`.)

Thus, with very few lines of code, a scalable, threaded, load-balanced algorithm has been implemented!

**ADVANCED CAPABILITIES AND FUTURE WORK**

In an ideal parallel algorithm, threads operate on data without retaining state between invocations, and they read from and write to shared memory without contention. In the example above, no initialization or final reduction is required on each thread, and writing to memory is uniquely determined by the point ID on which the thread is working.

In more realistic applications, however, many algorithms require synchronization, initialization, and maintenance of local state. To provide these capabilities, `vtkSMPTools` offers the following tools. (For more information, see the previously referenced document, “Parallel Processing with the SMP Framework in VTK.”)

1. Thread local storage

Thread local storage is used to store VTK instances and other objects. Developers can instantiate objects using `vtkSMPThreadLocal` and `vtkSMPThreadLocalObject` (for VTK classes that require VTK-specific allocation and deletion). These objects are created once per thread, and they persist as each edge processes its assigned tasks. Implementing such local thread objects is a great way to accumulate output over multiple thread invocations.

2. Functor initialization and reduction

Functor initialization and reduction is typically used in combination with the creation of local thread objects, since these objects must often be initialized. The results are combined (or reduced) into final form as execution concludes. `vtkSMPTools` provides these capabilities by invoking the optional `Initialize()` and `Reduce()` methods (defined by the functor in combination with the `operator()` method).

3. Atomic integers

Atomic integers can be used to ensure data integrity when reading from and writing to shared memory. `vtkSMPTools` supports a subset of the C++11 atomic API, mainly the operators `++`, `-`, `+=`, `-=`. These integers can be used when multiple threads must access and manipulate a common data structure.

Currently, many VTK algorithms are being rewritten with these simple constructs with good performance scaling. New algorithms are also being designed and implemented (both in the `vtkSMPTools` framework and in VTK-m). If you are interested in realizing performance gains in your own code, we recommend that you build VTK with `vtkSMPTools` enabled. (At the moment, TBB is the most popular.) We also look forward to receiving contributions of your new and old algorithms implemented using this framework.

**REFERENCES**


Will Schroeder is president and CEO of Kitware. His role is to identify technology and business opportunities and to obtain the necessary support for Kitware to meet these opportunities. Will also provides technical leadership for large open-source projects such as the National Library of Medicine Insight Toolkit project and VTK.

Sujin Philip is a member of Kitware’s Scientific Computing team. His background is in using massive and hybrid parallelism (combining message passing, shared memory multithreading, and GPGPU) for scientific computing and visualization applications.

Berk Geveci is the Senior Director of Scientific Computing at Kitware. He is one of the lead developers of the ParaView visualization application and VTK. His research interests include large-scale parallel computing, computational dynamics, finite elements, and visualization algorithms.
ANATOMY OF A PARAVIEW CATALYST PYTHON SCRIPT

Andrew Bauer (Kitware), Benjamin Jimenez (Army Research Laboratory), Rajneesh Singh (Army Research Laboratory)

INTRODUCTION

ParaView Catalyst has yielded some amazing results. Such results include iso-surfacing and generating images with 256K Message Passing Interface (MPI) ranks on Mira at Argonne National Laboratory, as well as computing particle paths and outputting iso-surfaced extracts at 42K cores using Adaptable Input/Output System (ADIOS). The wide variety of outputs that have been obtained with Catalyst can sometimes get lost with significant runs such as these. For example, Catalyst can capture data extracts and/or images of slices, streamlines, internal boundary surfaces, and a whole slew of derived quantities.

One of the key features of Catalyst is the ability to use Python in situ for specifying the analysis and visualization pipelines. Python is an amazing tool, as it provides flexibility while also simplifying the workflow. The decision to use Python was well thought-out during the initial design of ParaView’s in situ capabilities. Even though Python was not widely regarded as a “proper” high-performance computing (HPC) language, performance comparisons between pure C++ specified pipelines and Python-specified pipelines showed negligible performance differences. Basically, the performance penalty for using Python instead of C++ was less than a second for thousands of simulation time steps.

While the performance penalty of using Python is negligible, the flexibility it offers Catalyst users is a major feature. Catalyst users can tweak the desired output from a Python script and run a simulation without doing any recompiling. Inserting additional logic is simple as well.

Although the ParaView graphical user interface’s (GUI’s) Catalyst script generator is very powerful and can capture a wide variety of general Catalyst outputs, there will always be complex outputs that cannot be easily captured in the GUI. Examples of modifying Catalyst Python scripts to get desired, non-trivial outputs include the following:

- obtaining stereo-generated images from a Catalyst instrumented simulation code run
- moving a slice plane to examine the flow over a rotating helicopter rotor blade

Before getting into the details of a Catalyst Python script, we need to provide a bit of background on how Catalyst operates. A key point is that Catalyst will not output information at every time step. Accordingly, a two-step process occurs. The first step is to check if any of the Catalyst pipelines need to compute information at that point in the simulation run.

This is done through the RequestDataDescription() method in the Python script. If none of the pipelines have work to do, then control is returned to the simulation code with only a negligible amount of work done.

On the other hand, if one or more of the Catalyst pipelines need to execute, then (and only then) will the full set of VTK objects be created, which is used to represent the simulation’s grids and fields. After that, the DoCoProcessing() method in the Python script is called. RequestDataDescription() and DoCoProcessing() are the only two required methods in a Catalyst Python script. Both of them have a single argument, which is a vtkCPDataDescription object.

We will start dissecting the details of a Catalyst script with one generated from ParaView 4.3.1. It is a very simple pipeline where the data is loaded from a representative data set. Next, a slice filter is used, and finally, a writer is specified to output every 10th time step into a file, called slice_%t.pvtp. Note that the %t will be replaced by the time step.

The generated Python code looks like the following, except with comments, which have been removed for brevity:

```python
from paraview.simple import *
from paraview import coprocessing

def CreateCoProcessor():
    def _CreatePipeline(coprocessor, datadescription):
        class Pipeline:
            paraview.simple._DisableFirstRenderCameraReset()
            grid = coprocessor.CreateProducer(datadescription, \'input\')
            slice1 = Slice(Input=grid)
            slice1.SliceType = 'Plane'
            slice1.SliceOffsetValues = [0.0]
            parallelPolyDataWriter1 = \
                servermanager.writers.XMLPPolyDataWriter(\n                    Input=slice1)
            coprocessor.RegisterWriter(\n                parallelPolyDataWriter1, \n                filename='slice_%t.pvtp', freq=10)
        return Pipeline()
```

```python
def CreateCoProcessor():
    def _CreatePipeline(coprocessor, datadescription):
        class Pipeline:
            paraview.simple._DisableFirstRenderCameraReset()
            grid = coprocessor.CreateProducer(datadescription, \'input\')
            slice1 = Slice(Input=grid)
            slice1.SliceType = 'Plane'
            slice1.SliceOffsetValues = [0.0]
            parallelPolyDataWriter1 = \servermanager.writers.XMLPPolyDataWriter(\n                Input=slice1)
            coprocessor.RegisterWriter(\n                parallelPolyDataWriter1, \n                filename='slice_%t.pvtp', freq=10)
        return Pipeline()
```
class CoProcessor(coprocessing.CoProcessor):
    def CreatePipeline(self, datadescription):
        self.Pipeline = _CreatePipeline(self, \
        datadescription)

coprocessor = CoProcessor()
freqs = {'input': [10]}
coprocessor.SetUpdateFrequencies(freqs)
return coprocessor

coprocessor = CreateCoProcessor()
coprocessor.EnableLiveVisualization(False, 1)

def RequestDataDescription(datadescription):
    global coprocessor
    if datadescription.GetForceOutput() == True:
        for i in range(datadescription.\ 
        GetNumberOfInputDescriptions()):
            datadescription.GetInputDescription(i).\ 
            AllFieldsOn()
        return

coprocessor.LoadRequestedData(datadescription)

def DoCoProcessing(datadescription):
    global coprocessor

coprocessor.UpdateProducers(datadescription)
coprocessor.WriteData(datadescription);
coprocessor.WriteImages(datadescription, \ 
rescale_lookuptable=False)
coprocessor.DoLiveVisualization(\ 
    datadescription, "localhost", 22222)

REQUESTDATADESCRIPTION
The assumption made when constructing a Catalyst Python
script is that it will have a set of outputs that is needed at
specified time-step intervals. This works well for many simu-
lation codes.

The portions of the above Python script that take care of this
are as follows:

coprocessor.RegisterWriter(\ 
    parallelPolyDataWriter1, \ 
    filename='slice_%t.pvtp', freq=10)
...
freqs = {'input': [10]}
coprocessor.SetUpdateFrequencies(freqs)
...
def RequestDataDescription(datadescription):
    global coprocessor
    if datadescription.GetForceOutput() == True:
    for i in range(datadescription.\ 
    GetNumberOfInputDescriptions()):
        datadescription.GetInputDescription(i).\ 
        AllFieldsOn()
        datadescription.GetInputDescription(i).\ 
        GenerateMeshOn()
    return

coprocessor.LoadRequestedData(datadescription)

Here, the designated 10 output frequency is specified
twice. The first specification occurs when registering
the writer so that the script knows it needs to output
slice_%t.pvtp every 10th time step. The second speci-
fication is in the freqs dictionary, which is used in the
RequestDataDescription() method to specify whether
or not this pipeline needs to perform any computation.

Additionally, GetForceOutput() is used in situations
where the simulation code knows something important is
happening and can force the pipeline to output the slice
data, regardless of the time step. Typically, this is done at the
beginning or end of a simulation run.

For some simulations, however, this is not ideal. For example,
in a rotorcraft simulation run, it is difficult to apply appro-
priate initial conditions. Therefore, the first portion of the
simulation is computed only to reach a reasonable starting
point for analysis. In this case, the above code snippet can
be modified to ignore outputting anything from Catalyst for
the first 1,000 time steps, unless the output is forced.

This looks like the following:

def RequestDataDescription(datadescription):
    global coprocessor
    if datadescription.GetForceOutput() == True:
        for i in range(datadescription.\ 
        GetNumberOfInputDescriptions()):
            datadescription.GetInputDescription(i).\ 
            AllFieldsOn()
        return
    if datadescription.GetTimeStep() < 1000:
        return

coprocessor.LoadRequestedData(datadescription)

For simulation codes that have drastically varying time-
step lengths, the time-step value is likely not appropriate
for outputting information from Catalyst. In such cases, it
may be more beneficial to use the simulation time to create
the logic for when a pipeline should execute. This informa-
tion is available through the datadescription object's
GetTime() method.
DOCOPROCESSING

In the Python script shown above, the Catalyst pipeline is created when the script is imported. The `DoCoProcessing()` method updates the pipelines at requested points in the simulation. To non-experienced Python developers, this part of the code can be slightly confusing, so we will go into greater detail.

The pipeline begins with the `grid = coprocessor>CreateProducer(datadescription, 'input')` line, which corresponds to the reader in the ParaView GUI's pipeline. The line may seem overly complex, but for some simulations, there can be multiple inputs, so this is used to disambiguate them. For example, in climate simulations, the pipeline's source may be desired in either its true geometry (i.e., an oblate spheroid) or in a projection, depending on the desired Catalyst computation. With the source of the pipeline created, the slice filter's properties are set in the exact same way as in ParaView's Python interface.

The following part of the pipeline does this:

```
slice1 = Slice(Input=grid)
slice1.SliceType = 'Plane'
slice1.SliceOffsetValues = [0.0]
```

We can make the pipeline more complex either through the GUI or by modifying the code, but we will not go into that here. For users interested in learning more about advanced pipelines, we suggest reading *ParaView Catalyst User's Guide Version 2* [1] and the online ParaView Python application programming interface (API) documentation [2].

The next lines in the code create the writer and specify the desired output file name and frequency:

```
parallelPolyDataWriter1 = \servermanager.writers.XMLPPolyDataWriter( 
  Input=slice1)
coprocessor.RegisterWriter( 
  parallelPolyDataWriter1, 
  filename='slice_%t.pvtp', freq=10)
```

As noted earlier, this pipeline is created when the Python script is imported and then executed, as needed, during the simulation run. For the rotorcraft simulation previously mentioned, it would be nice to be able to modify the location of the slice plane during the simulation run. To accomplish this, we simply need to modify the `slice1` object's properties in the `DoCoProcessing()` method. Assuming we have methods defined that obtain the slice's normal and origin (`getslicenormal()` and `getsliceorigin()`, respectively), we can modify the `DoCoProcessing()` method to update that information while the simulation code is running.

```
def DoCoProcessing(datadescription):
    global coprocessor
    coprocessor.UpdateProducers(datadescription)
    coprocessor.Pipeline.slice1.SliceType.Origin = 
        getsliceorigin(datadescription)
    coprocessor.Pipeline.slice1.SliceType.Normal = 
        getslicenormal(datadescription)
    coprocessor.WriteData(datadescription);
    coprocessor.WriteImages(datadescription, 
        rescale_lookuptable=False)
    coprocessor.DoLiveVisualization( 
        datadescription, "localhost", 22222)
```

RESULTS

A Catalyst Python script very similar to what is shown above was used with the Computational Research and Engineering for Acquisition Tools and Environments - Air Vehicles (CREATE-AV™) Helios [3] rotorcraft simulation code to output the flow field over a cross-section of one of the rotating blades. The images on the following page are examples created from the moving slice plane. Note that while the blade stays in the middle of the image, the axis widget at the bottom-left corner of the image is rotating, delineating slice rotation with the rotor blade.

To see additional example images, please watch the video "Helicopter Rotor Flowfield," which is available on Kitware's Vimeo page (https://vimeo.com/126419999).

CONCLUSIONS

While the ParaView GUI's Catalyst script generator can capture complex pipelines, it would be impossible to capture all desired Catalyst output behavior through the GUI. Even if this was attempted, it would make the GUI too complex. Instead, Catalyst relies on the flexibility of Python to provide specialized output that is not easily captured in the GUI.

In this article, we demonstrated how the Catalyst output can be customized for a rotorcraft simulation code with a couple of lines of Python. The two additions to the script allowed skipping wasteful output during the simulation startup steps and modifying the slice plane configuration to follow the rotation of the helicopter blade.

In a similar manner, data scientists can customize Catalyst scripts to easily capture important outputs that would otherwise be overly expensive to obtain. The net result is a more flexible *in situ* visualization analysis tool that speeds up the analysis workflow and can be implemented to efficiently use HPC resources.

For additional information on Catalyst and its capabilities, including a webinar and the user's guide, please visit http://www.paraview.org/in-situ/.
The above images show the slice-plane output from a CREATE-AV™ Helios simulation run.

ACKNOWLEDGEMENTS

Material presented in this article is a product of the CREATE-AV™ Element of the CREATE Program sponsored by the U.S. Department of Defense HPC Modernization Program Office.

REFERENCES


Andrew Bauer is a research and development engineer on the Scientific Computing team at Kitware. He primarily works on enabling tools and technologies for HPC simulations.

Benjamin Jimenez has several research interests, including external vehicle aerodynamics using computational fluid dynamics, as well as rotorcraft aerodynamics and structural dynamics (CFD+CSD) coupling.

Rajneesh Singh leads a team of engineers and scientists in the Vehicle Technology Directorate (VTD) of the U.S. Army Research Laboratory (ARL) at Aberdeen Proving Ground in Maryland.
The implementation of numerical simulations has led to several benefits for the scientific community, including cost and time savings, improved product performance, and new product design and manufacturing. With high-performance computing (HPC), the potential benefits are significantly greater [1]. For years, the Visualization Toolkit (VTK) and ParaView development teams have been dedicated to providing state-of-the-art techniques that can be used to post-process results from HPC simulations, both visually and analytically. Visualization and analysis are only part of the simulation workflow, however, as depicted below.

The workflow begins with defining the appropriate shape and topology of the geometric domain. This involves combining various domain descriptions including computer-aided design (CAD) models, discrete geometry, and spatially-sampled data such as Light Detection and Ranging (LiDAR) data and computerized tomography (CT) data.

The next step is to define the simulation by specifying material properties and initial boundary conditions on the geometric domain. Any information tied to the specific numerical solver also needs to be specified. For example, it may be necessary to define the type of numerical preconditioner that will be used, along with the type of solver, the start time, the number of time steps, and the desired accuracy.

Typically, the geometric domain is discretized into a set of geometrically simpler shapes (such as triangles, quadrilaterals, tetrahedra, and hexahedra) through mesh generation to support the underlying numerical formulation. The simulation information is then transferred from the geometric model to the resulting mesh. Together, the information and the mesh define the input to the simulation. The simulation is run, and the results are analyzed and visualized. Based on these results, the geometric model can be changed to meet a desired target.

**CMB FRAMEWORK**

In 2008, Kitware received funding from the U.S. Army Engineer Research and Development Center (ERDC) to build a set of tools to support hydrological simulation workflows. As part of this effort, Kitware created Computational Model Builder (CMB). CMB is a framework that enables various toolkits and modeling systems to be used to form an end-to-end solution for supporting simulation life cycles.

Unlike other frameworks that provide similar functionality, CMB integrates with external packages such as simulation codes, geometric modeling kernels, and mesh-generation tools. This approach supports complex workflows using the many powerful simulation tools developed throughout the industry. For example, CMB interfaces with the analysis by exporting model, mesh, and attribute information into a format supported by the simulator. CMB is also highly customizable. Not only can CMB be tailored through its plugin architecture, but parts of CMB's graphical user interface (GUI) can be tailored through an XML-based file description, and parts of CMB that are accessible through Python can be customized through Python scripting.

CMB leverages several open-source toolkits and frameworks such as VTK, ParaView, MoleQueue [2], and Qt. Since 2007, CMB has been expanded to support a wide variety of simulation domains, including nuclear energy. As part of the CMB effort, two toolkits that address key requirements of the simulation life cycle have also been developed. One is Remote Meshing/modeling Utilities (ReMUs) [3], which provides access to meshing technology and remote modeling systems.

*The above image shows a mesh of a nuclear reactor generated using ReMUs and applying tools in MeshKit, which is developed by Argonne National Laboratory.*
The other is Simulation Modeling Toolkit (SMTK) [4], which provides access to geometric and topological modeling functionality and represents necessary simulation attribute information. CMB, ReMUs, and SMTK are being developed by Kitware under a BSD license.

**CMB MODELBUILDER**

One of the first applications to be released based on CMB is ModelBuilder. ModelBuilder allows users to load in geometric models, view their topologies, and access modeling operations supported by the underlying native modeling system through SMTK’s model interface. This flexible interface is available for a variety of modeling kernels, including the following:

- parametric kernels – volumes, faces, and edges have well-defined parametric coordinates that are good at representing curved models
- discrete kernels – volumes, faces, and edges are each discretized into a piecewise collection of primitive cells that may have internal parameterizations but do not guarantee a parameterization at the model-entity level

ModelBuilder can display a model’s boundary representation (BRep), and users can control display properties associated with elements within the BRep. ModelBuilder currently supports these types of modeling sessions:

- Common Geometry Module (CGM) – a code library that grants access to parametric CAD models (both Open CASCADE and ACIS models, provided you have access to the ACIS kernel software)
- Discrete Model – a discrete model representation developed at Kitware
- Exodus – a specific discrete model representation based on the Exodus file format

**SUPPORTING SIMULATION INFORMATION**

ModelBuilder provides access to SMTK’s simulation attribute management system and allows users to describe the various types of information that need to be defined for each simulation, as well as how that information needs to be associated with part of the geometric model. In both ModelBuilder and SMTK, this information is represented by a set of attribute objects.

An attribute object consists of a collection of item objects (strings, integers, doubles, files, etc.) that can model complex, hierarchical data structures. The attribute subsystem provides validity checking based on an item object’s constraints. Attribute objects can be associated with the appropriate model entities in an SMTK model and retrieved when writing out the appropriate input decks for a specific simulation system. Attribute objects can also be used to represent information related to a specific solver, such as pre-conditioners and time steps.

The structure of the simulation information can be easily defined in an XML file using a simple text editor. When this attribute “template” file is loaded into ModelBuilder, the application automatically generates a GUI based on the defined requirements. The GUI-generation process can be extended by creating custom views, which are incorporated using CMB’s plugin architecture.

The above contains examples of complex Open CASCADE models displayed using SMTK.

The above is an attribute template file and resulting GUI. It is important to note that the same mechanism used to model simulation information in CMB is used to present and specify modeling operations such as Booleans (intersections, unions, and differences) and edge/face splits and merges.
For example, an initial discrete model may have a single model face representing the Chesapeake Bay area. This area is shown in the image below. In order to properly model the geometric domain, the areas that form both Chesapeake and Delaware bays need to be separated from the offshore area and the channel that joins the bays in the north. Describing a face-splitting operation using an attribute object to indicate the face to be split can do this. In the image, face splitting is demonstrated by the different colors: Chesapeake Bay is green, Delaware Bay is orange, and the offshore area is blue.

The above image illustrates the creation of a discrete model from Digital Elevation Model (DEM) data of the Chesapeake Bay. The DEM data is courtesy of ERDC.

In the case of a meshing operation, information describing the mesh sizing field and the types of mesh elements to be generated can also be represented by attribute objects.

**PYTHON EXPORTING**

When a user creates an attribute template, he or she also defines a Python script file. The script generates simulation input files from attribute, model, and mesh resources stored in ModelBuilder. When the simulation description is complete, the user can invoke the associated script via ModelBuilder’s interface and generate the simulation’s files. Any missing information detected by the exportation process is relayed back to the user so that it can be addressed.

**SIMULATION EXAMPLE**

In the following image, ModelBuilder is used to define a computational fluid dynamics (CFD) simulation of the flow around a pin in a nuclear reactor. The simulation is performed with Hydra, an extensible multiphysics code. The example shows the pin, along with a spacer grid that forces coolant mixing to occur around the pin. The model is colored to depict the various model faces that make up the geometric domain. The image also displays one of the attribute-driven GUI panels, which is defined by an attribute template file designed for Hydra.

The next image shows the simulation results visualized in ParaView, based on the input files generated by ModelBuilder.

For more information on CMB, please visit http://www.computationalmodelbuilder.org.

**ACKNOWLEDGEMENTS**

This material is based upon work supported by the Department of Energy under Award Number DE-SC0007615. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service
Kitware is developing software extensions that aim to address complex search problems common in fields such as security and defense as part of the Defense Advanced Research Projects Agency (DARPA) Memex program.

A key use case of the technology is the ability of law enforcement to discover and address human trafficking through the Web. The prominence of websites that facilitate trafficking is increasing, necessitating new software platforms that address the scale and scope of the expanding Web. Current software and search approaches do not effectively integrate interactive and social media, text, images, and video, while taking into account the degree of importance of each piece of media, which is required for performing “deep searches.”

"The leading search providers are singularly focused on displaying ads, which has led to closed systems and stagnant interfaces," Jeff Baumes, a Co-Principal Investigator for the project and Technical Lead at Kitware, said.

Kitware is collaborating with NASA’s Jet Propulsion Laboratory at the California Institute of Technology and Continuum Analytics LLC to create specific extensions to Apache Nutch with the goal of improving its overall technological superiority for search. The project, which was featured on 60 Minutes earlier this year, is funded by DARPA. The work seeks to integrate state-of-the-art algorithms developed by Kitware with multimedia and video analytics to provide image and video understanding support to allow for automatic detection of objects and massive deployment via Nutch.

"The objective of our project is to create rich, customizable search experiences, using open-source tools and integrating diverse content such as video and images," Baumes said.

The project is designed to create an interactive and visual interface for initiating Nutch crawls. The plan for the interface is to use the Blaze platform to expose Nutch data and to provide a domain-specific language for the crawls. The interface is also tasked with using the Bokeh visualization library to deliver simple interactive visualization and plotting techniques for exploring crawled information. Moreover, the team intends to make improvements to media-oriented search, as well as to the user interface (UI) and the domain-specific language for search, to unleash “deep search” activities that can be easily implemented by law enforcement and analysts for quick turnaround in time-critical situations.

"Our motivation is to enable rapid search and visualization in ways that were not possible in the past," Baumes said.

**REFERENCES**


**Bob O’Bara** is an Assistant Director of Scientific Computing at Kitware. His main areas of interest include geometric modeling, model and mesh visualization, mesh generation, and general scientific visualization techniques.

**David Thompson** is a research and development engineer at Kitware. His interests include conceptual design, solid modeling, computational simulation and visualization, and mechatronics.

**Andrew Bauer** is a research and development engineer on the Scientific Computing team at Kitware. He primarily works on enabling tools and technologies for HPC simulations.

**Yumin Yuan** is a research and development engineer on the Scientific Computing team at Kitware. He is the main software architect of the suite of CMB applications that regard geometric modeling, meshing, pre- and post-simulation analysis, and visualization.

**John Tourtellott** is a research and development engineer at Kitware. His areas of focus include software tools for scientific computing, digital map visualization, and user interface design.

**Robert Maynard** is a research and development engineer at Kitware. He is an active contributor to VTK and CMB. His interests include HPC using accelerators, message brokers, and build systems.
In Phase II, the team will develop an extendable, turn-key platform that is capable of fully-automated, high-throughput electron tomography of nanoscale materials. The platform will feature a modern user interface and enhanced graphical tools for editing, aligning, segmenting, visualizing, and analyzing data. It will also feature one-click tomography, making it possible to easily reconstruct experimental data and visualize it in 3D.

“As a result of the project, it will be possible to have the entire tomography process, from measurement to 3D analysis, take place on a single graphical platform,” Hanwell said. “This technology will greatly accelerate progress in materials characterization and metrology, while also promoting reproducible science.”

For additional information, please read the full announcement on our news site.

This material is based upon work supported by the Department of Energy under Award Number DE-SC0011385.

TEAM MEMBERS ATTEND CVPR 2015

Kitware was actively involved this year in the Institute of Electrical and Electronics Engineers (IEEE) Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), which took place in June. An annual event, CVPR draws over 2,000 attendees from academia and industry, providing them with opportunities to learn more about the field of computer vision through workshops, tutorials, and conference sessions. Many of Kitware’s computer vision experts participated in this year’s event, and Kitware was a conference sponsor.

Members of Kitware hold live booth demonstrations.

On the first day of the conference, Matt Leotta from Kitware joined Sameer Agarwal from Google, Pierre Moulon from FOXEL SA, Frank Dellaert from Georgia Tech, and Vincent Rabaud from Aldebaran - Softbank Group to present “Open Source SfM.” The presentation highlighted Kitware’s Motion-imagery Aerial Photogrammetry Toolkit, known as MAP-TK. MAP-TK, which is open source, provides estimations of camera flight trajectory and sparse 3D point clouds of a scene. The presentation was well attended, and it concluded with one attendee winning a Parrot Bebop Drone.
Later in the week, Kitware presented a poster on "Collaborative Computer Vision R&D" at the fourth annual Vision Industry and Entrepreneur Workshop, which was held in conjunction to CVPR. The poster highlighted details on Kitware's computer vision expertise including key capabilities, customers, and collaborators, as well as information on the KitWare Image and Video Exploitation and Retrieval (KWIVER) toolkit. KWIVER is Kitware's open-source computer vision platform. It tackles challenging image and video analysis problems, addressing various domains worldwide.

Members of Kitware were also invited to discuss "Visual Profiling: Social Media and Surveillance for Commercial Intelligence" at the IEEE Workshop on Large Scale Visual Commerce (LSVisCOM). The discussion focused on information that can be applied directly to worldwide events that occur on a regular basis.

**PROJECT REPOSITORIES MIGRATE TO GITLAB**

Kitware is now using GitLab to host several of its open-source projects, including the Visualization Toolkit (VTK), ParaView, Computational Model Builder (CMB), and Simulation Modeling Toolkit (SMTK). GitLab helps unify development workflows across various projects, providing the ability for community members to contribute and review changes, such as bug fixes and new features.

Additional advantages to migrating project repositories to GitLab include a more accessible user interface (UI) for inspecting changes and posting comments, as well as a simplified code-review process, especially for topic branches with multiple commits.

To access Kitware's project repositories on GitLab, please visit https://gitlab.kitware.com/explore, and for questions regarding the migration, please e-mail the VTK, ParaView, CMB, or SMTK mailing list.

**OPEN-SOURCE FRAMEWORK AIMS TO ENABLE NEW PARADIGMS IN STRUCTURAL DESIGN**

Kitware has announced Department of Defense (DoD) funding to develop an open-source, distributed computational environment for virtual materials exploration to address a pressing need in the manufacturing community. Currently, manufacturers that heavily leverage advanced materials engineering are limited in their abilities to efficiently and inexpensively design precision parts. As a result, there is a great need for new paradigms in structural design that leverage best-of-breed tools from finite element analysis (FEA) and materials simulation codes.

"There now exist simulation codes that are able to predict materials properties with increasing levels of accuracy," Marcus Hanwell, the project's Principal Investigator and a Technical Leader on Kitware's Scientific Computing team, said. "Adopting more advanced materials simulation would ultimately enable the structural design process to drive materials requirements, with the possibility of real-time materials exploration, composite design, and/or processing options that could be adjusted alongside conventional optimization strategies such as varying shape to accommodate load/rigidity requirements."

The open-source framework will enable designers to move beyond materials as fixed design inputs to active variables that can be manipulated as part of the structural design process. This has the potential to result in a paradigm shift from whole-part design based on standard material properties tables to the use of actual variables in dynamic, active material design. The framework will also allow designers to explore materials and compositions in real time and to adjust parameters to accommodate specific requirements.

"Once designers begin to incorporate advanced materials models, it becomes possible to tailor designs to increase longevity or reduce the weight of a part without compromising structural integrity," Hanwell said. "For example, advanced designs might specify advanced materials processing techniques such as heat treatments in specific regions to increase local performance without the added cost of manufacturing the entire part out of an expensive material."

To advance materials simulation, Kitware's experts in finite element modeling (FEM) and FEA will research and prototype a multiscale simulation environment that integrates advanced materials models with a classical FEA system for modeling complex systems. For this Phase I Small Business Innovation Research (SBIR) project funded by the DoD, the team will investigate different approaches using software as a rapid prototyping/research tool to critically assess the benefits and costs of each approach.

For additional information, please read the full announcement on our news site.

This material is based upon work supported by the United States Air Force under Award Number FA8650-14-M-5044.

**NATURE METHODS FEATURES IN VIVO CELL CYCLE PROFILING PAPER**

The paper "In vivo cell cycle profiling in xenograft tumors by quantitative intravital microscopy" was published in the June edition of Nature Methods. It presents an integrated work-
flow for automated in vivo cell cycle profiling that enables large-scale, unbiased, consistent, and reproducible studies of the spatiotemporal effects of chemotherapeutic drugs and experimental perturbations on cell-cycle progression in living tissues at a single-cell level. The workflow combines a high-resolution intravital imaging setup for longitudinal observation of cell fate in living tissues with a novel computational framework for automated 3D segmentation and cell-cycle state identification of individual cell nuclei with widely varying morphologies, even those embedded in a highly complex tumor environment.

The paper is authored by Kitware’s Deepak R. Chittajallu, along with Stefan Florian, Rainer H. Kohler, Yoshiko Iwamoto, James D. Orth, Ralph Weissleder, Gaudenz Danuser, and Timothy J. Mitchison. It is also available on http://www.nature.com.

KITWARE PLACES ON TECHNOLOGY LIST

Kitware ranked sixth on the Albany Business Review’s "The List: Technology Service Companies," which was released in the paper’s Tech Pages edition. The list ranks companies by number of employees in the Capital Region.


EMPLOYMENT OPPORTUNITIES

Kitware is seeking talented, motivated, and creative individuals to fill open positions and join in its mission to develop and deliver state-of-the-art software products and services. In particular, Kitware is seeking to hire highly-skilled research and development engineers to join its Scientific Computing team to work on developing and improving leading solutions in visualization software. Several additional positions are available on Kitware’s Scientific Computing, Medical Computing, and Computer Vision teams.

The impact of your research at Kitware will extend far beyond the organization, as Kitware’s open-source business model will allow you to become part of the worldwide communities that surround Kitware’s projects. Kitware employees enjoy a collaborative work environment that empowers them to pursue new opportunities and to challenge the status quo through novel ideas.

In addition to providing an excellent workplace, Kitware offers comprehensive benefits including: flexible hours; a computer hardware budget; health, vision, dental, and life insurance; short- and long-term disability insurance; immigration and visa processing; a relocation bonus; tuition reimbursement; and a generous compensation plan.

For additional information, please visit Kitware’s employment website at http://www.jobs.kitware.com. Interested applicants are encouraged to submit their resumes and cover letters through the online portal.

KITWARE INTERNSHIPS

Kitware internships provide current college students with the opportunity to gain training and hands-on experience working with leaders in their fields on cutting-edge problems. Interns assist in developing foundational research and technology across Kitware’s five solution areas.

Contributors: Lisa Avila, Andrew Bauer, Berk Geveci, Benjamin Jimenez, Robert Maynard, Bob O’Bara, Sujin Philip, Will Schroeder, Rajneesh Singh, David Thompson, John Tourtellott, and Yumin Yuan

Graphic Design: Steve Jordan
Editor: Sandy McKenzie

This work is licensed under an Attribution 4.0 International (CC BY 4.0) License.

Kitware, ParaView, CMake, KiwiViewer, and VolView are registered trademarks of Kitware, Inc. All other trademarks are property of their respective owners.