This fall, we’re thrilled to highlight releases of some of our newest projects. We announced the debut of VeloView, an open-source project for visualization of LiDAR data, and the next release of Bender, a toolkit for anatomical modeling.

Fall also brings a busy conference season, and preparations are in full swing. Our technical team will be attending, presenting, and participating in a range of events, including TEDxAlbany; Innovation, Design, and Emerging Alliances in Surgery (IDEAS): Virtual Surgery; and the IEEE International Conference on Computer Vision. We’ll also be exhibiting at SC13 in Denver, CO at booth 4207! You can find more details on our SC13 participation in the Kitware News section.

For our full calendar of events, please see the Kitware events page at www.kitware.com/events. If you would like to set up a time to meet with us at any of these events to discuss employment opportunities, potential collaboration, or consulting services, please contact us at kitware@kitware.com, and we would be happy to arrange a meeting.

VELOVIEW 1.0 RELEASED

In September, Kitware and Velodyne LiDAR released VeloView 1.0, an open-source application designed for real-time, 3D visualization and analysis of point cloud data generated by Velodyne’s 3D LiDAR sensors HDL-32E and HDL-64E. The application provides an easy-to-use tool for visualization and processing of Velodyne’s data-rich, 3D LiDAR sensors, supporting live sensor streams and run-time display of previously captured data.

VeloView ingests data from the HDL sensor, which sweeps an array of lasers 360 degrees at 10 hz and captures more than one million points per second. VeloView displays the laser measurements as point cloud data and supports custom color maps of multiple variables such as intensity-of-return, time, distance, azimuth, and laser id. VeloView also ships with a plugin that makes network and file readers available inside ParaView so that users may take advantage of the full range of visualization and data analysis capabilities in ParaView.

VeloView includes a Python scripting engine, which makes it easy to extend and customize. With the Python console in VeloView, users can access point cloud data and attribute arrays and use NumPy to perform advanced data analysis.

For more information or to get started with VeloView, visit the ParaView download page and set Type of Download to Community Contributed Applications.
CMAKE 2.8.12

CMake 2.8.12 was released in October. The CMake development team would like to thank the CMake users who reported issues during the CMake 2.8.12 release cycle; your contributions are greatly appreciated.

The CMake 2.8.12 release features several significant developments and enhancements, including the new compile option commands, alias targets, optimized custom command dependency lookup, generate stage file creation, and XCode 5 support. There are also three new CMake policies, CMP0021-CMP0023. For details on these, please see the release blog post on the Kitware blog.

The new commands in the 2.8.12 release improve the usability of CMake and make it more efficient for developers. The target_compile_options command allows developers to specify compile to use when compiling a given target. This command supports PUBLIC, PRIVATE, and INTERFACE options. Further, the add_compile_options command allows developers to add options to the compiler command line for sources in the current directory and below. Both of these new commands support generator expressions.

BENDER 1.1 RELEASE

Kitware is pleased to announce the release of version 1.1 of Bender, with binaries available on Midas and source code under git version control.

Bender is an open-source toolkit based on 3D Slicer that provides algorithms and a user-friendly application for rigging, skinning, posing, and volume resampling, enabling researchers to interactively reposition an anatomic labelmap into a desired pose.

The 1.1 release features a number of enhancements, notably usability improvements. Algorithms perform faster and consume less memory, and some corner cases (for example when hands are in contact with the hips) are better handled. The workflow has been redesigned to be more streamlined, with the number of clicks greatly reduced. Upcoming developments to Bender will improve the anatomical repositioning by modeling tissue-specific deformation characteristics more precisely. For an introduction to Bender features, there is a tutorial with sample data available on the Bender wiki.
With the trends of supercomputing, this workflow will not fully realize the goal of quick turnaround for gaining valuable insights from numerical experiments. The numerical experiments will run quicker due to the increase in computing power. However, the bottleneck is moving from running the actual simulation to post-processing the results to gain insight into the problem at hand. The main issue is dealing with the data efficiently. This classical workflow involves writing a large amount of the simulation data to disk just to read it back in again to do post-processing. For the situation where the post-processing is done on a separate machine, at best it has only one magnitude less computing power than the supercomputer the data was computed on exacerbating the issue. If this separate machine doesn’t share a disk with the supercomputer then this involves another file transfer which again slows down the workflow.

**ENABLING IN SITU ANALYSIS AND VISUALIZATION**

Our solution to this problem is to perform visualization and analysis on the supercomputer as the simulation is running. This has a variety of names including *in situ*, co-processing and co-visualization. There are many advantages to this. The first is that we can begin the analysis and visualization process without having to do any file IO for the simulation results. The second is that we have the full compute power of the supercomputer available to do this processing. Additionally we expect that writing the results from the analysis and visualization will be much smaller than the original full simulation data. Generally, the desired results can be saved in a much more compact form than the original data such as streamlines, lift on a wing, maximum stress, etc. This leads to dealing with simulation-generated data that is in a much more compact form, and also has the benefit of not requiring significant extra post-processing in order to gain insight into the problem at hand.

**PARAVIEW CATALYST FOR IN SITU COMPUTING**

While there has been significant work on performing *in situ* analysis and visualization, our goal was to create a general library that could easily be used to analyze and visualize data in a variety of ways. If the user can do their post-processing with ParaView they can use Catalyst for co-processing. ParaView and VTK have already been demonstrated to be successful tools for post-processing analysis and visualization for a wide variety of simulations. Catalyst has been a relatively new addition to ParaView to enable the same analysis and visualization tools that are already available for post-processing to be provided for *in situ* processing. Additionally, one of the design goals of ParaView Catalyst was to make it very simple for simulation users to create customized outputs in a workflow that they are already familiar with. This is done through ParaView’s CoProcessing plugin. In this new workflow, instead of dealing with the large data sets output from a simulation code the user specifies their desired output during the pre-processing stage by using this plugin. They essentially create the pipelines to process the results in their desired manner. At the end of the co-processing pipelines, instead of writing out data extracts (e.g. contours, streamlines, etc.) and/or screenshots, the user specifies individual frequencies for outputting these data extracts and/or screenshots. All of this information is saved in a Python script.

When the simulation is run with Catalyst, these pipelines will be executed at the requested time steps. Note that each sink in the pipeline can have an independent output frequency. For the adventurous, these Python scripts can be edited to contain more sophisticated logic to better extract desired data.

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

![Figure 2: ParaView Catalyst enabled workflow.](image)
PARAVIEW CATALYST DETAILS

The easiest way to get access to ParaView Catalyst is through a regular ParaView build. By default, the CMake option to enable the API to Catalyst is turned on (the option is PARAVIEW_ENABLE_CATALYST). Typically, Catalyst is also built with MPI and Python enabled, although neither are required to use the Catalyst libraries. Additionally, with the next release of ParaView (version 4.1) we will also be releasing “editions” of Catalyst. The purpose of these editions is to reduce the overhead of linking to Catalyst. While ParaView and VTK contain a wide variety of analysis and visualization tools, a Catalyst-enabled simulation run will typically use a very small subset of those tools. The rest of the tools can result in a fairly large overhead in libraries that a simulation code needs to link to. For example, if no screenshots are desired from Catalyst then it doesn’t make sense to include all of the rendering and compositing parts of ParaView and VTK plus their dependencies (e.g. an OpenGL implementation). Our experience to date has shown that we can reduce the library size by over 100 MBs by using specific Catalyst editions for simulation runs. While this may not seem like a large amount by itself, considering the fact that multiple copies of the libraries will need to be loaded for a large simulation run, the numbers can add up quickly. For static executables, each MPI process will store a full copy of the executable. For executables that use dynamic libraries, only a single copy of a shared library per node is needed but each shared library needs to be loaded into memory which can severely burden the IO system. For a supercomputer like Hopper at NERSC with 6,384 compute nodes and 24 cores per node, this would correspond to over 600 GBs of extra storage for executables with shared libraries or 15 TBs of extra storage for static executables when running at full scale.

INTERFACING A SIMULATION CODE TO PARAVIEW CATALYST

In order for a simulation code to use Catalyst for co-processing, it must store the simulation data in a format that ParaView understands. Essentially, this means that the simulation data must be stored using VTK’s data model. This information is beyond the scope of this article but is covered thoroughly in the ParaView Catalyst User’s Guide (http://paraview.org/Wiki/images/4/48/CatalystUsersGuide.pdf). The design that was used for ParaView Catalyst leverages an adaptor that bridges the information in the simulation code’s data structures and VTK’s data structures. This is shown in the figure below.

![Diagram of the Catalyst design](image_url)

**Figure 3: Design to bridge between the solver and Catalyst.**

This design isolates the simulation code’s dependency on Catalyst such that it doesn’t need to include any VTK or ParaView classes in that code base. Additionally, Catalyst doesn’t need to be modified to gain access to the simulation code’s data structures. The adaptor is responsible for passing in situ details to Catalyst (e.g. the Python scripts, simulation time step, etc.) and translating the simulation code’s data structures to VTK data structures. The interface between the simulation code and the adaptor is normally done through function calls. Typically there are only a handful of calls to pass the proper information to the adaptor. These are:

- Initialization call: This call initializes both the adaptor and Catalyst. This is where the co-processing pipelines are set up in Catalyst.
- Processing call: This call checks to see if any co-processing needs to be performed. If a pipeline needs to execute, the adaptor is responsible for creating/updating the VTK data structures from the current simulation code state so that the pipelines can execute with that data.
- Finalization call: This call cleans up any state in the adaptor and/or Catalyst.

As can be seen with this design, the functional interface allows a minimal code footprint inside of the simulation code codebase.

For more information on ParaView Catalyst, visit the website at http://catalyst.paraview.org/.

ACKNOWLEDGEMENTS

Initial funding for ParaView Catalyst development was through Army SBIR contract W91126-08-C-0062. ParaView Catalyst was developed collaboratively between Sandia National Laboratories, Los Alamos Laboratory, the Army’s Aeroflightdynamics Directorate, and Kitware. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Andrew Bauer is an R&D Engineer in the scientific computing team at Kitware. He primarily works on enabling tools and technologies for HPC simulations.
For engineers and architects, it is essential to understand and communicate how the structure (cantilever, truss girder, frame, etc.) they are designing behaves under loading. The Sketchalyze app, built on top of VTK[1] / VES[2] and available in the App Store for free, aims to support professionals and students in this process by letting them sketch structural systems quickly and by providing deformations and internal forces of structural members immediately. The direct and intuitive way users can interact with the structural system on a tablet device makes it easy to try new ideas and encourages exploration of structural behavior by experimenting with geometry, loading, and dimensions.

The Sketchalyze app (Figure 1) provides tools to sketch a structural system in a way that is familiar to people who have used a drawing program before. A finite element computational core computes deformations and section forces on the device and common techniques such as vector warp, mapped colors, and line plots are applied for visualization. Whenever the system is modified, analysis results are recomputed, and the visualization is updated accordingly. The finite element analysis for small to medium sized structures is fast enough to provide a realtime feedback when a user manipulates the system.

From the beginning of the project, it was clear that structures should be rendered in a full 3D volumetric representation and a choice regarding the related technology had to be made. The release of VES just before the Sketchalyze project got underway made for an easy decision, as VES leverages the visualization power of VTK.

BACKGROUND
VES (VTK for Embedded Systems) is an open-source framework for accelerating mobile visualization on current generation mobile hardware. VES consists of two core libraries: VES and Kiwi. The VES library provides rendering capabilities and infrastructure for scene management by utilizing OpenGL ES 2.0 [3]. The Kiwi library is an application framework built on top of VES and VTK. Figure 2 shows the technology stack of VES based application to depict the relationship between different components.

![Figure 2: Stack diagram for VES based applications. Kiwi provides a GLUE layer between the VES core and VTK. VES supports OpenGL and OpenGL ES 2.0.](image)

In the next section, we have described the implementation of Sketchalyze, which leveraged and extended VES for its own requirements. We are hoping that some of these new functionalities will be released in the next version of VES.

IMPLEMENTATION
The finite element core of Sketchalyze has been designed by the authors for the requirements of a highly interactive application. The interface between the finite element subsystem and the visualization subsystems is mostly based on function objects and local coordinate systems. Function objects encapsulate the mathematical functions which represent geometry (deformed and undeformed) as well as the computed member forces (normal force, shear force, bending moment). Most of the time, these functions are polynomials or piecewise polynomial. In the mechanical model of a structure, each member has an attached local coordinate system. The mechanical theory of one-dimensional continua like trusses or beams is based on a local coordinate system attached to the reference line of the member. For visualiza-
tion purposes, this coordinate system can be conveniently used to transform point coordinates between local and global reference frames; our corresponding class provides the respective methods.

Several subclasses of vtkAlgorithm have been implemented in order to translate our domain-specific information into vtk-PolyData objects to be processed in the various pipelines for the individual model aspects.

**SOURCES AND FILTERS**

Most fundamental is the ElementSource class which provides the vtkPolyData for the detailed representation of structural members. It uses the local element coordinate system (either in undeformed or deformed configurations) and transforms the vertices of the section to global coordinates (see Figure 3 top). In order to select a visually appealing representation at structural joints, all members connected to one node are investigated, and simple heuristic rules determine the choice between different drawing options (cut off, draw with capping, extend, draw rounded).

In structural analysis, it is common to draw the distribution of section forces as plot along the element length (see Figure 3 bottom). The corresponding vtkPolyData object is provided by the LinePlotSource class which also makes extensive use of the element’s local coordinate system.

**Figure 3: Sket analytic structural models: Detail of a beam structure and distribution of bending moments in a suspended frame under line loads.**

Structural supports are visualized by symbols reminiscent of historical bridge bearings. Figure 3 (top) shows two constraint symbols which allow for a rotation about exactly one axis, the left one can move horizontally, the other one does not allow translations. Generally, in three dimensions, there are $2^6 = 64$ (3 translations, 3 rotations) types of supports. The visual representation is generated by a vtkGlyph3D in conjunction with our generic ConstraintSymbolSource algorithm class which has 64 output ports, one for each support type. The input for the glyph is generated by a ConstraintSource class specific to our finite element model which tries to orient symbols such that they don’t overlap with elements.

Further developments include classes for generating visual representations of line loads and forces. Note the non-uniform scaling of force symbols in Figure 3 (top).

**HYBRID ACTORS**

Textual annotations on top of the 3D representation are crucial, e.g. in order to include numerical values in a concise fashion. While VTK provides a wide range of 2D actors, this feature is not yet present in VES. However, it was surprisingly simple to include hybrid 2D drawing for iOS. We introduced a vesActor2D class which draws on a glass pane on top of the GLKView. The view class maintains a list of its 2D actors and lets them draw when necessary. The following snippet of Objective-C++ code shows the stripped down drawing (some coordinate handling is omitted) method of a vesActor2D subclass which displays labels for scalar point data.

```cpp
(void)drawRect:(CGRect)rect inView:(vesView *)view {
    double p[3], pD[3];
    vtkPolyData *pd = ...; // get input polydata
    vtkPoints *points = pd->GetPoints();
    vtkDataArray *data = pd->
    GetPointData()->GetScalars();
    for (int i = 0; i < points->
    GetNumberOfPoints(); i++) {
        NSString *s =
            [NSString stringWithFormat:@"%f", data->
            GetTuple1(i)];
        points->GetPoint(i, p);
        worldToDisplay(view, p, pD);
        [s drawAtPoint:CGPointMake(pD[0], pD[1])
            withAttributes:nil];
    }
}
```
USER INTERACTIONS

There exist various alternatives to implementing user interactions in a 3D software system. For instance, the vesKiwiWidgetRepresentation lets subclasses handle gestures. Simplicity is one of the main advantages of this approach. On the other hand, it may lead to difficulty in maintaining code in complex software projects since it becomes very easy to mix application logic into view objects. For this reason, we decided to handle interactions at the UIView level. Gestures are recognized and forwarded to the active tool object which interprets the raw input and translates it into modifications of the finite element model.

Different tools let users create elements; move nodes or elements; and add and manipulate forces or loads etc. The most important class in this context is the WorkingPlane. It converts the coordinates of a point from view to world coordinates, handles snap-to-grid, and is able to locate model objects. This picking capability is currently implemented by a linear search in the model, and for the model size targeted by the Sketchalyze app.

PERFORMANCE CONSIDERATIONS

Performance is crucial for an app like Sketchalyze. Only if models can be created and manipulated fluently, is a good user experience guaranteed. Currently, for a mid sized structure with about 50 members, the app operates at a rate of around 5 frames per seconds (depending on the type of section etc.) on an iPad 3 including the finite element analysis and the generation of approx. 10000 polygons of polydata. Although we still have continued development to do, we have tackled several iterations in analyzing and improving performance bottlenecks to produce Sketchalyze 1.0.

Other than initially expected, it turned out that the finite element analysis is not critical for the overall performance (as long as a linear analysis is performed). The amount of time needed for the analysis of a small structure with 50 members is around 0.04 seconds on an iPad 3.

Most relevant for performance is the element source which generates several thousands of polygons and points. By default, normals and triangles are generated by VTK filters and then the data is handed over to VES. Therefore, the first step in improving the performance was to include the normals generation into our ElementSource class. It now generates triangle strips and normals, thus improving the overall performance.

Another crucial piece of code is the conversion of coordinates from the element frame to the global system. After experimenting with functions from Apple’s Accelerate framework (which contains a BLAS implementation), it turned out that it is most efficient to unroll the loops and type things out. Further performance improvements might be achieved using SIMD instructions on NE-ON.

ACKNOWLEDGEMENTS

The Sketchalyze app started as Master’s thesis of Duy Thanh Truong and has been supported by the Vietnamese-German University, the University of Bochum and the University of Kassel.

REFERENCES


Matthias Baitsch is interim professor for structural analysis at the University of Kassel, Germany. He has a background in civil engineering with a focus on design optimization and numerical methods.

Duy Thanh Truong studied Computational Engineering at the Vietnamese-German University (Ho Chi Minh City, Vietnam) and received his M.Sc. from Ruhr-University Bochum (Bochum, Germany). He is the main developer of Sketchalyze and currently working as research assistant at the University of Kassel (Germany).

Aashish Chaudhary is an R&D Engineer on the Scientific Computing team at Kitware. Prior to joining Kitware, he developed a graphics engine and open-source tools for information and geo-visualization. Some of his interests are software engineering, rendering, and visualization.

DID YOU KNOW THAT USING VES AND KIWI, YOU CAN BUILD YOUR OWN ANDROID OR IOS APPS?

For Android devices, developers can use CMake and the Android Native Development Kit (NDK) to cross-compile VTK and the VES/Kiwi libraries for Android. Developers can further enhance their apps by designing Java Native Interface (JNI) libraries. Similarly, on iOS, they can use CMake and the iOS SDK to cross-compile the libraries. With the KiwiViewer Android or iOS app, developers can then package assets, load data, and render geometry.

To learn more about VES and Kiwi for building Android apps, visit the Kitware webinar page to view the recorded Android tutorial webinar.

For help designing a custom visualization app tailored to your needs, please contact us at kitware@kitware.com.
Laparoscopic surgery is a minimally invasive surgical approach in which instruments are passed through ports placed at small incisions. Typically, a surgeon will use her own medical intuition and external landmarks on the patient to select the placement of the surgical ports through which to pass the surgical instruments. However, this port selection strategy can fail if the surgeon lacks experience, or the internal anatomy does not correspond to the external landmarks. In these cases, the surgeon finds that the pre-selected set of ports is not suitable for reaching a surgical site; the surgeon must then choose between making even more incisions for more ports, or abandoning the laparoscopic approach to perform a more invasive procedure.

A PORT PLACEMENT VISUALIZATION MODULE FOR 3D SLICER
To help address this challenge, Kitware is developing a Port Placement module for the medical imaging tool, 3D Slicer. This module can assist in port placement planning by visualizing some ports and simulated surgical tools over preoperative medical imagery. The visualization allows the surgeon to see the result of a given selection of ports during preoperative planning; for instance, the surgeon might see that a simulated surgical tool cannot reach a surgical site without passing through a vital sensitive organ, so the surgeon can interactively adjust the port position until the visualization shows that the tool has unobstructed access to the surgical site. This functionality can lead to port placement planning that can help ensure that the surgeon will have the flexibility required to safely and successfully perform the surgical procedure.

This is implemented as a scriptable module, meaning it was implemented in Python and does not require compilation. The module implements all of its functionality using preexisting 3D Slicer modules. A set of ports is specified as a list of fiducial nodes, which can be easily added to a scene with a click of the mouse. The surgical tools at each port are simulated as long, thin cylinders. Each tool can either be manually oriented within the module using the intuitive Transforms widget, or the surgeon can also specify a surgical target and click the “Aim Tools at Target” button to automatically orient all tools toward the target. If the surgeon wants to adjust a port’s placement, she can click and drag its corresponding fiducial marker and the simulated surgical tool will follow the mouse to the new position.

The Port Placement visualization module is available as a Slicer extension for Slicer 4.2.

AUTOMATED PORT PLACEMENT FOR ROBOTIC LAPAROSCOPIC SURGERY
Laparoscopic surgeons have gained even greater dexterity, accuracy, and vision at surgical sites via the assistance of robotic systems like Intuitive Surgical’s da Vinci System. However, there are unique challenges to surgical port placement planning for robotic surgery. For instance, it is difficult for a human to judge how much dexterity the robot will have at a surgical site due to complexity of the robot’s kinematic structure. It may seem that the robot can aptly perform a task through a given surgical port, but during the procedure it may turn out that the task is just out of the robot’s reachable workspace, or near a singularity (a robotic configuration where its degrees of flexibility are decreased). Furthermore, in order to move surgical tools inside the patient’s body, the robot must move its arms outside the patient. This means that, in addition to the usual challenges of port placement planning, the surgeon must also ensure that the robot’s arms won’t collide with each other or the rest of the operating room during the procedure.

To assist with these issues, Kitware is also developing an easy-to-use system that can recommend a suitable port placement plan for a given robotic surgical procedure. The surgeon inputs a procedure represented by a set of 6-D task frames of surgical interest that the robot tools need to reach, as well as a set of points to serve as candidate positions for surgical ports. The port placement plan returned by our procedure includes the positions of the surgical ports on the patient’s body, the robot arm configurations to reach these ports, and the position of the robot’s base on the operating room floor. This system will provide an intuitive interface to the surgeon for specifying the intended surgical task, and
the system will choose a port placement plan that optimizes the surgeon’s dexterity at the surgical site.

The interface used by the surgeon to specify the surgical task will be implemented as a module for 3D Slicer. Surgical task frames will be specified using fiducial markers (markers currently support position and orientation, but orientation is not yet accessible via the Slicer GUI) and candidate port positions will be specified as curves or surfaces using control points (also with fiducial markers). The surgeon can load the patient’s preoperative medical imaging into 3D Slicer in order to visually guide their placement of surgical task frames and candidate port positions.

The automated port placement recommendation system is based on cutting-edge work in automated surgical planning [1]. The system uses knowledge of the robot’s kinematic structure and the surgeon’s requirements to return a plan that (1) reaches all specified surgical task frames, (2) maximizes the robot’s dexterity at those frames, and (3) maximizes the plan’s robustness to uncertainty in the surgical procedure due to noisy medical imaging and variability in patient anatomy.

The automated port placement system requires fast computation of the robot’s forward and inverse kinematics; we implemented the kinematics routines using the Eigen C++ linear algebra library [2]. Eigen provides a MATLAB-like interface to a wide variety of highly efficient linear algebra routines. The port placement system also requires performing a complex nonlinear optimization with many nonlinear constraints. We implemented the optimization using the open source nonlinear optimization library NLOpt [3]. The NLOpt library offers easy-to-use interfaces to a wide variety of cutting-edge local and global nonlinear optimization routines; for the port placement problem we use Constrained Optimization by Linear Approximations (COBYLA) [4] because it does not require specification of an analytical derivative. Currently, our automated port placement system can return a plan to reach 3 surgical task frames within 5 seconds.

We are currently adding support in our planning system for the third endoscopic arm of the da Vinci System. We are also developing the Slicer module that the surgeon will use for intuitive specification of the required surgical procedure. We soon will have a full port placement planning system that can assist surgeons by reducing surgical setup time and lowering the likelihood of misplaced ports - and scars caused by extra port incisions - in robotic laparoscopic surgery.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Hamidreza Azimian for his invaluable insight on port placement planning for robotic laparoscopic surgery. This work is supported by NA-MIC and the 3DSlicer community. This work was partially supported by NSF GRFP Grant No. DGE-1144081 and NIH/NIBIB Grant No. 1R43EB014074-01.

REFERENCES


Luis Torres is a Ph.D. student at the University of North Carolina at Chapel Hill, where he conducts research in medical robotics. His recent focus has been on image-guided motion planning for snake-like robotic needles.

Andinet Enquobahrie is a Technical Leader at Kitware. Andinet is responsible for technical contribution and management of image guided intervention and surgical simulation projects. His recent efforts are focused on use of PET-CT imaging to improve the clinical effectiveness of lesion biopsy, laparoscopic surgical procedures and tools for image-guided intervention application development and bioinformatics analysis.
Kitware and NYU-Poly have been working together on the development of open-source tools and libraries to make it easier for developers to provide sophisticated, interactive geoinformatics over the web. Kitware and NYU-Poly, along with help from climate scientists from National Labs and NASA, are collaborating on ClimatePipes [1] project, the goal of which is to enable climate data access, analysis, and visualization over the web for masses. The work performed for ClimatePipes has resulted in front-end libraries written primarily using JavaScript with minimal dependencies such as JQuery and D3. For the backend, we use Python as our programming language because of its popularity, easy of use, and availability of bindings to VTK [2], UVCDAT [3], and many other computing and analysis libraries. Currently, the backend library uses VTK and UVCDAT primarily for data processing and analysis.

In the next few sections, we will provide an overview of some of these libraries with details on some of the core features currently supported by them.

**GEOJS: GEOVIS LIBRARY FOR THE WEB**

As the name suggests, the GeoJS library is developed using JavaScript and provides the high-level API for geoinformatics over the web. GeoJS’s primary functionality is to deliver WebGIS using the latest Web 2.0 technologies such as 2D Canvas, WebGL [4], and HTML 5.0. For the most part, the API provided by GeoJS should look familiar to developers with a WebGIS or traditional GIS background. GeoJS provides a layer-based API in which a layer can contain one or more geometries. The layers are arranged in a stack layout in which the layer on top of the stack is drawn last by the renderer. Figure 2 shows the technology stack for GeoJS.

GeoJS is built on top of VGL, the rendering library of ClimatePipes. VGL provides a high-level API for scene construction and management. Most of the implementation for VGL is borrowed from our previous work on VES [5]. As of release 0.1 of GeoJS, the following core features are supported:

- Render order based on the position of the layer in the stack
- Animations for a one or more layers
- Map layer using OpenStreetMap datasets
- Data query / picking using mouse clicks
- Workflows API and editor

Apart from these core features, many other essential features are supported. Currently GeoJS supports reading data in the GeoJSON format, and we are planning to add support for reading Shapefile in the near future. In this article, we will focus on two of the core features of GeoJS: workflows and provenance.

**Workflows and Provenance**

Two of the important features supported by GeoJS are workflows and provenance. The backend for workflows and provenance uses the VisTrails [6] workflow and provenance management software to perform the analysis and data operations needed by the front end. The workflows primarily have two components. In the backend, running
the workflow through VisTrails performs computation and algorithmic operations on the data. In the front end, GeoJS provides a web-based workflow editor to visualize and interact with the workflows.

The workflow component of the GeoJS library contains classes to draw and interact with a workflow. The workflow is drawn on the 2D context of an HTML5 canvas using basic drawing primitives. Unlike the VisTrails workflow editor, inputs exist directly in the modules, allowing users to set and change parameters of the workflow. These inputs are standard html input elements that hover over the canvas in the DOM. The workflow editor consists of a list of modules on the right, the main workspace to the left of the modules list, and a few buttons along the bottom. Modules can be dragged from the list on the right into the main workspace area, which instantiates a new instance of that module. Initiating a drag operation from an output port of one module to an input port on another creates connections between modules. Modules can be selected by a mouse click, upon which the module gets highlighted with a yellow border. Clicking and dragging a module moves it about the canvas. Clicking and dragging on the blank canvas pans the view. Selecting them and clicking the delete button can delete modules.

As mentioned earlier, GeoJS uses VisTrails python API at the backend. A trimmed down version of the open-source VisTrails workflow management system is used to execute GeoJS-generated workflows on the server side. VisTrails workflows can be exported and imported as XML. Using this functionality enable us to generate workflows on the front end in JSON, send them to the server, serialize them as XML, import them into VisTrails, execute them, and send the results back to the Web client. Any libraries that can run on VisTrails can be utilized by our system in this way.

VGL: OPENGL VISUALIZATION LIBRARY
VGL is developed in response to the interactive rendering requirements of ClimatePipes. We chose to develop VGL to deliver a lightweight, extensible, and open-source library for enabling developers to build high-performance, interactive visualization applications on the web clients. Most of the existing libraries are either focused on games and animations or lack the performance required to render large dataset on the web browser. As shown in Figure 4, the VGL library is composed of multiple components, each of which offers a unique feature within the library.

SceneGraph and Rendering Pipeline
The VGL library uses scene graph data structures to manage scenes efficiently. A scene graph is a data structure that
provides spatial and logical relationships between various entities of a scene. A scene graph can be implemented in many ways, and some of the open-source implementations of a scene graph are inspired by the design of OpenGL Performer, one of the well-known scene graph libraries from SGI [7]. The VGL library is built using the same core principles and additionally provides a consistent, easy-to-use API to allow applications to take advantage of programmable pipeline functionality of WebGL.

Like VES, VGL separates the geometry from the material. A material defines the look and appearance of the geometry. Since it’s desirable to share same material between different geometries, VGL provide the material API on the vglNode and not on vglMapper since a node (for example a vglGroup-Node) can have one or more mappers via child actor nodes (vglActor). In this way a material can be shared between different mappers and the rendering can be grouped together by the material used by the node in order to minimize OpenGL state changes for maximum performance.

In WebGL, it is necessary to provide a vertex and a fragment shader in order to render geometry primitives (See Figure 6). Vertex shaders can be used for traditional, vertex-based operations such as transforming the position with a matrix, computing the lighting equation to generate a per-vertex color, and generating or transforming the texture coordinates. The fragment shader is a general-purpose method for interacting with fragments. VGL provides a consistent API for applications to pass vertex and fragment shaders to the programmable pipeline. Also, as shown in Figure 5, the pipeline requires uniforms and attributes to be passed to the shaders. VGL hides all the complexities of the OpenGL pipeline and provides an easy to use API for both uniforms and vertex attributes.

Another feature delivered by VGL is its flexible, OpenGL-friendly geometry data structure for maximum performance and portability. VGL provides a very flexible data structure for defining geometry for the purpose of rendering. Some of the highlights of the VGL library geometry data are:

- Support for interleaved or separated data arrays
- Any number of coordinate systems for the point data
- Support for different basic types for the point data
- Separation of point data from the cell data
- Extensible data structure

VGL geometry data structure is composed of one or more sources (arrays). These sources when combined with primitive types define attributes of the vertices such as positions, colors, or user defined attributes.

**Figure 6. Simple overview of WebGL programmable pipeline**

**CONCLUSION & FUTURE WORK**

In this article, we presented an overview of the GeoJS and VGL library for the purpose of visualization and analysis over the web that uses UVCADT, VisTrails, and VTK on the backend. We are extremely delighted by results despite some challenges with developing and testing large code base using JavaScript. We are hoping to further improve the code base, adding new features, testing the code thoroughly, and releasing the source code frequently as part of the ClimatePipes project.

**ACKNOWLEDGEMENTS**

Many thanks to Dean Williams, Charles Doutriaux from LLNL, Berk Geveci from Kitware, Claudio Silva from NYU-Poly, Thomas Maxwell and Gerald Potter from NASA for their support and encouragement.
For the sixth consecutive year, Kitware is on the Inc. 5000 list, spurred on by our successful open-source business model and collaborative work. The Inc. 500|5000 list is an exclusive ranking of the nation’s fastest-growing private companies.

“Making the Inc. 5000 list for six years in a row is a testament to the excellent people at Kitware and their dedication to the work we do,” said Will Schroeder. “Kitwareans are contributing and making an impact in a variety of open-source and technical communities, and our growth has empowered us to explore new opportunities.”

In a stagnant economic environment, Kitware was able to grow competitively and create new jobs at both our New York and North Carolina offices, as well as open a new office in New Mexico.

**NEW AWARD TO IMPROVE AIR FORCE VIDEO ANALYST WORKFLOWS WITH VIRAT**

Kitware received a new broad agency announcement (BAA) award from the Air Force Research Laboratory to integrate our Video and Image Retrieval and Analysis Tool (VIRAT) functionality with the agency’s Planning & Direction, Collection, Processing & Exploitation, Analysis & Production, and Dissemination experimentation environment (PCPAD-X). Integration of these two tools will improve analyst performance in exploiting Full Motion Video (FMV).

FMV exploitation is currently very labor intensive, and often involves teams of people monitoring a single video feed to extract the value for the primary mission. There are a number of drawbacks to this approach as it requires analysts to monitor the video feed and “call out” events for another analyst to record. The monitoring analysts focused on finding mission-specific elements of interest may overlook other events and objects of interest to other parties. In addition, the communication of events between shifts of analysts on a particular mission, or to outside entities, is cumbersome and may result in non-standard, fragmented descriptions of events and other salient information.

We have developed a powerful FMV exploitation capability that addresses a number of drawbacks to the current process. The VIRAT tool, developed primarily under DARPA funding, automatically processes FMV in real-time to extract and record elements of interest, including moving targets and events, or objects that match a specific appearance model.

**REFERENCES**


---

**Aashish Chaudhary** is an R&D Engineer on the Scientific Computing team at Kitware. Prior to joining Kitware, he developed a graphics engine and open-source tools for information and geo-visualization. Some of his interests are software engineering, rendering, and visualization.

**Ben Burnett** is a senior software developer in the vgc research group at NYU Poly. He received his M.S. in Computing from the University of Utah where he focused on Data Management and Analysis.

**Chris Harris** is an R&D Engineer at Kitware. Chris’s background includes middleware development at IBM, and working on highly-specialized, high performance, mission critical systems.

**David E. DeMarle** is a member of the R&D team at Kitware where he contributes to both ParaView and VTK. He frequently teaches Kitware’s professional development and training courses for these product applications.

**Daniel Kohler Osmari** received his B.S. in Computer Science from Universidade Federal do Rio Grande do Sul (UFRGS) working with GPU programming; he’s currently a M.Sc. student at Polytechnic Institute of New York University (NYU-Poly) working on information visualization and high performance computing.
For this project, VIRAT will be integrated with PCPAD-X to extend its capabilities to increase analyst productivity, provide an improved user interface and more standardized communication and recording mechanisms, and include a state-of-the-art object tracking system for automatically detecting unusual events in the data using normalcy models.

This work is supported by the Air Force Research Laboratory (AFRL) under Contract No. FA8750-13-C-0107. The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official AFRL position, policy, or decision unless so designated by other documentation.

EXHIBITING AT SUPERCOMPUTING 2013

From November 18-21, Kitware will be exhibiting at Supercomputing 2013 in Denver, Colorado. Supercomputing is the leading conference for the high-performance computing (HPC) community. At this year’s event, Kitware will be exhibiting at booth 4207, giving several presentations and participating in the tutorial session.

As part of the weekend tutorial series, there will be a "Large Scale Visualization with ParaView" Tutorial on Sunday, November 17 from 8:30 am - 12:00 pm. The tutorial will be taught by Dave DeMarle, W. Alan Scott, Li-Ta Lo, and Kenneth Moreland. This tutorial presents the architecture of ParaView, an open-source turnkey application for analyzing and visualizing large data sets in parallel, and the fundamentals of parallel visualization. Attendees will learn the basics of using ParaView for scientific visualization with hands-on lessons. The tutorial features detailed guidance in visualizing the massive simulations run on today's supercomputers and an introduction to scripting and extending ParaView.

On Tuesday, as part of the “Parallel Universe” set of talks at the Intel Theater (booths 2501 and 2701), Kitware will give a short talk on large-scale data visualization and analysis. Following up on that talk, as part of the Exhibitor Forum Series on Wednesday, Utkarsh Ayachit will present a talk titled "Tools for Data Analysis and Visualization at Exascale." This talk will provide greater detail and insight following the Intel presentation. It will focus on new approaches to visualization and analysis that will scale to the next generation of computers.

This session will highlight the Visualization Toolkit (VTK) and ParaView, the leading open-source tools for scientific visualization and analysis. It will also cover the collaborative efforts on the Data Analysis at Extreme (DAX) toolkit and Portable Data-Parallel Visualization and Analysis Library (PISTON), and how to leverage these next-generation paradigms with VTK and ParaView to move your work to the exascale. Further, the session will detail the business benefits of basing a custom solution on open-source tools, such as avoiding vendor lock-in and leveraging a community's software maintenance skills.

We will also be showcasing and demonstrating some of our latest work at the Kitware booth. The demos will include interactive marching cubes using DAX; interactive browsing of large histology databases over the web; and immersive, six-degree-of-interaction with scientific data sets in ParaView. A schedule of the demos and material will be posted on the Kitware website closer to the show.

Be sure to stop by the Kitware booth to learn more about our current cutting-edge work, discuss our current employment opportunities, and pick up some Kitware swag!

U.S. AIR FORCE RESEARCH LAB AWARD TO EXTEND BENDER TOOLKIT

Kitware received a new Phase I SBIR award from the U.S. Air Force Research Laboratory to advance the Bender toolkit. The project aims to reduce the time required for anatomical model pose manipulation and for simulating anthropomorphic changes, such as an increase in body-mass index (BMI).

The Bender improvements will be a multi-stage process, beginning with extending the toolkit to make use of freely available libraries of pose and motion data. Next, the user interface will be broadened to support the specification of changes in regional fat or muscle volumes so that the BMI of the model can be changed. Ultimately, real-time, GPU-based surgical simulation methods will be integrated to speed the computation of realistic anatomic dynamics.

The Phase I effort will include replacing the dual-quaternion skinning capabilities already implemented in Bender with Kitware's multi-grid solver in order to improve speed and reduce memory requirements while also providing more accurate motion models. This work will build upon the linear and non-linear finite element modeling (FEM) solvers in the “Simulation Open Framework Architecture” (SOFA) toolkit for surgical simulation.

The primary result of this Phase I exploration and the planned Phase II development effort will be an intuitive program for performing voxelized anatomical model rigging, manipulation, and resampling.

This work is supported by the Air Force Research Laboratory under Contract No. FA8650-13-C-6444. The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Air Force position, policy or decision unless so designated by other documentation.

NIH PHASE II SBIR AWARD TO DEVELOP OPEN-SOURCE DIGITAL PATHOLOGY SYSTEM

Kitware recently won a new $1M Phase II SBIR from the National Institutes of Health to enhance and further deploy its open-source digital pathology system. On the heels of a successful Phase I SBIR project, the Kitware team and our collaborators will improve their digital pathology system.
and web portal to facilitate a large pathology network, bringing pathology into the digital realm already enjoyed by other medical specialties and scientists. The inexorable push toward digital data acquisition has not yet taken hold in pathology, but with this open-source system, pathologists will see benefits such as easy annotation and markup of whole slide images, flexible data archival and sharing, telepathology, 3D visualization and informatics, and education.

As part of the Phase II effort, new development will be focused on the architecture and web portal, formalizing an analysis framework and related tools, and generating new digital pathology content for both educational and reference purposes. The data-centric architecture will provide the foundation for data ingestion and processing of whole slide images (WSI), and support web-based display on a variety of devices including mobile devices. With that foundation, the team will develop a web portal that will enable users to upload, share, search, and view WSI. The web portal will also be the basis for the Digital Pathology Journal that will enable dissemination of results, including large and distributed images with annotations.

"We are excited to bring this capability to pathologists and drive the field forward," said Dr. Charles Law, Principal Investigator on the project. "We have already worked with six institutions to deploy the system for teaching purposes, and are thrilled by the results. The residents who have used the system have found it intuitive to use and say that it enriches their learning experience; they can readily collaborate with each other, and ultimately, will be able to work with pathologists throughout the country."

Research reported in this publication was supported by the National Institute Of Biomedical Imaging And Bioengineering of the National Institutes of Health under Award Number R44EB013518. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

**NEW AWARD TO DEVELOP ADVANCED SPARSE BUNDLE ADJUSTMENT SYSTEM**

Kitware announced new Phase I SBIR funding from the U.S. Air Force Research Laboratory to develop innovative software for performing sparse bundle adjustment for full motion video (FMV) and wide-area motion imagery (WAMI).

Sparse bundle adjustment (SBA) is a technique for refining 3D scene geometry and the parameters of relative camera motion using images taken from different viewpoints. This technique has existed since the 1950s, but it is only in the past decade that the computational resources required to perform large-scale sparse bundle adjustment have become available and practical to use.

Existing open-source toolkits for SBA are slow and focus on large collections of unordered images. Significant optimizations can be made for SBA by exploiting properties found in aerial video, particularly temporal continuity. These optimizations are available at all stages of the camera calibration pipeline, from feature detection and matching to SBA itself.

With this new Phase I funding, Kitware will develop an SBA system targeted for aerial video that will produce state-of-the-art camera calibration accuracy in a fraction of the time compared to current unordered image collection software. A variation of the techniques developed under this project can also be used to construct a streaming SBA solution for live video. Furthermore, relaxation of the specific constraint that bundle adjustment accuracy be optimal over all images could enable real-time, streaming video feeds onboard military vehicles for intelligence, surveillance, and reconnaissance (ISR) systems.

To achieve these goals, Kitware will leverage its world-renowned computer vision expertise, algorithms, and open-source libraries to perform feature detection, feature matching, and sparse optimization.

"We're excited for the opportunity to improve a well-known yet under-utilized functionality and make a significant impact on video analyst workflows," said Matthew Leotta, Principal Investigator on this project. "An additional benefit is that our newly-developed system will build upon open-source software and our developed SBA toolkit will also be released to the public as open source."

This work is supported by the Air Force Research Laboratory under Contract No. FA8650-13-M-1694. This contract was approved on September 5, 2013 with FY 13 funds. The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official AFRL position, policy, or decision unless so designated by other documentation.
NEW EMPLOYEES
Scott Wittenburg
Scott Wittenburg joined our Santa Fe, NM office as an R&D Engineer. He studied at the University of New Mexico, where he earned his B.S. and M.S. in Computer Science. Scott has extensive experience as a consultant developer, working on a variety of projects, including an NSF Partnerships for Innovation grant to do calibration and real-time warping of HTML5 Video and Canvas applications when projecting onto arbitrary, non-planar surfaces.

Dan Lipsa
Dan Lipsa joined Kitware as an R&D Engineer at our Clifton Park, NY office. Dan earned his Ph.D. in computer science from Swansea University in the UK. His thesis was on the visualization of foam simulation data. Prior to joining Kitware, Dan worked as Senior Software Engineer for the Ecora Corporation in Portsmouth, New Hampshire.

Tristan Coulange
Tristan Coulange joined our office in Lyon, France as an R&D Engineer. He qualified as an informatics and research engineer in Centrale Lille (France) in 2012, then extended his scope of expertise with an Informatics Master at Lyon University. The Informatics Master specialized in image treatment and algorithmic geometry.

INTERNSHIP OPPORTUNITIES
Kitware Internships provide current college students with the opportunity to gain hands-on experience working with leaders in their fields on cutting edge problems. Our business model is based on open source software—an exciting, rewarding work environment.

In addition to providing readers with updates on Kitware product development and news pertinent to the open source community, the Kitware Source delivers basic information on recent releases, upcoming changes and detailed technical articles related to Kitware’s open-source projects.

For an up-to-date list of Kitware’s projects and to learn about areas the company is expanding into, please visit the open source pages on the Kitware website at http://www.kitware.com/opensource/provensolutions.html.

A digital version of the Source is available in a blog format at http://www.kitware.com/source.

Kitware would like to encourage our active developer community to contribute to the Source. Contributions may include a technical article describing an enhancement you’ve made to a Kitware open-source project or successes/lessons learned via developing a product built upon one or more of Kitware’s open-source projects. The Kitware Source is published by Kitware, Inc., Clifton Park, New York.

Contributors: Lisa Avila, Matthias Baitsch, Andrew Bauer, Ben Burnett, Aashish Chaudhary, David E. DeMarle, Andinet Enquobahrie, Chris Harris, Luis Torres, Robert Maynard, Daniel Kohler Osmari, and Duy Thanh Truong.

Graphic Design: Steve Jordan
Editor: Katie Osterdahl

This work is licensed under a Creative Commons Attribution 3.0 Unported License.

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