Depends: Workflow Management Software for Visual Effects Production

Andrew Gardner* Jonas Unger
Linköping University, Sweden

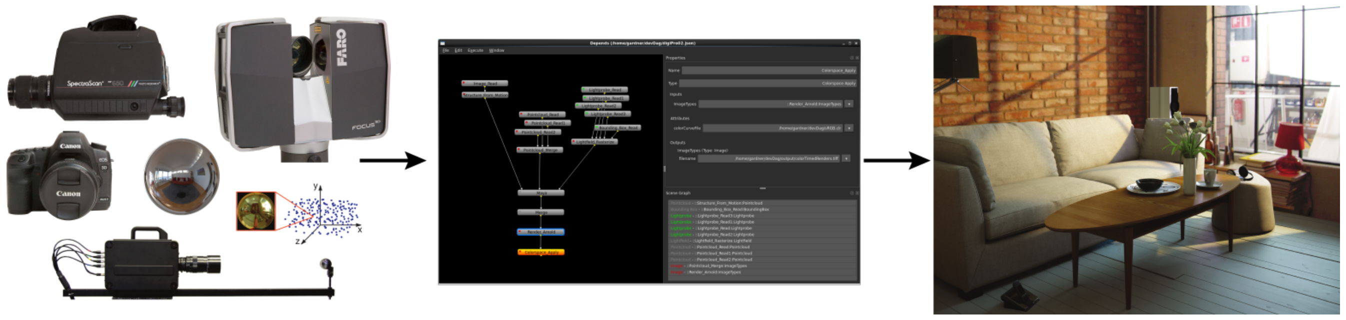


Figure 1: *Multiple capture devices (left) are used to scan a real-world photo studio. The resulting data and processing software is organized by the Depends workflow management application (center), to create a computer graphics rendering of a real-world environment (right).*

Abstract

In this paper, we present an open source, multi-platform, workflow management application named Depends, designed to clarify and enhance the workflow of artists in a visual effects environment. Depends organizes processes into a directed acyclic graph, enabling artists to quickly identify appropriate changes, make modifications, and improve the look of their work. Recovering information about past revisions of an element is made simple, as the provenance of data is a core focus of a Depends workflow. Sharing work is also facilitated by the clear and consistent structure of Depends. We demonstrate the flexibility of Depends by presenting a number of scenarios where its style of workflow management has been essential to the creation of high-quality results.

CR Categories: I.3.8 [Computer Graphics]: Applications—; H.4.1 [Information Systems Applications]: Office Automation—Workflow management;

Keywords: visual effects pipeline, directed acyclic graph, user interface, artist workflow

1 Introduction

Visual effects production environments are an ever-changing whirlwind of complex dependencies. The number of individuals working together and the level of detail required for a photorealistic film sequence are staggering, and the amount of data generated to achieve photorealism continues to grow. Artists must create content quickly and efficiently, as turnaround times are often short. These challenges require intense coordination and nimble, efficient operation from the studio as a whole, as well as from each individual within the pipeline.

A single effect, rig, or environment may consist of dozens or hundreds of different elements brought together to form a final look. Multiple processes may be used in coordination to generate these elements. Also, elements are rarely complete; new camera angles with which to view the data may expose flaws in the approved look, or tastes may simply evolve. The ability to quickly change an element's appearance is as important as the ability to revert to an old look. This ever-present uncertainty ensures that individuals working in a visual effects production environment must be organized and efficient.

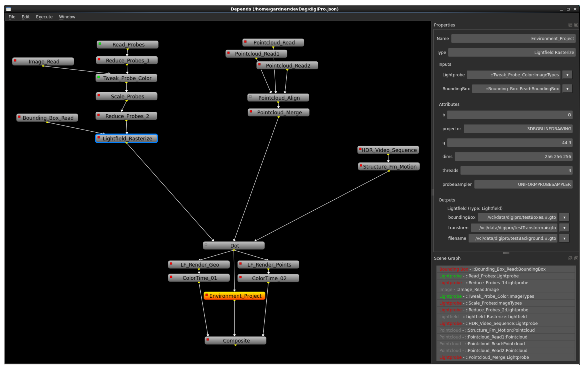
To this end, we have developed an open source workflow management tool named *Depends*, tuned specifically to address the needs of visual effects artists. As with many workflow managers, Depends strives to maintain ease and clarity of use. It does this by providing an interface to organize nearly any command-line application into a directed acyclic dependency graph (DAG), concurrently maintaining a list of generated data as a scenegraph. It is lightweight and extensible, with a focus on scalability. It includes functionality to preserve the provenance of data, allowing users to quickly recall configurations and artistic decisions long after the work has been done. Depends is developed using the Python programming languagge and the open-source PySide bindings for the Qt windowing toolkit. It is therefore highly portable and simple to run.

2 Workflow Management Systems

Workflow management systems, and the methods they employ, exist in many computing contexts. Tools that build dependency graphs to synchronize a system's components, such as the UNIX *make* build system, have historically proven effective. Various aspects of the UNIX philosophy such as modularity, composition, and separation [Raymond 2003] have made shell scripts a useful workflow management tool, but the complexities of fast-paced visual effects environments often necessitate software that adds organizational tools on top of established methods.

Katana, the software package released commercially by The Foundry, is an example of a workflow management application that

*e-mail:{andrew.gardner, jonas.unger}@liu.se



Properties

Name: Environment_Project
Type: Lightfield Rasterize
Inputs
Lightprobe: ::Tweak_Probe_Color:ImageTypes
BoundingBox: ::Bounding_Box_Read:BoundingBox
Attributes
b: 0
projector: 3DRGBLINEDRAWING
g: 44.3
dims: 256 256 256
threads: 4
probeSampler: UNIFORMPROBESAMPLER
Outputs
Lightfield (Type: Lightfield)
boundingBox: /vcl/data/digipro/testBoxes.#.gto
transform: /vcl/data/digipro/testTransform.#.gto
filename: /vcl/data/digipro/testBackground.#.gto

Scene Graph

```
Bounding Box - ::Bounding_Box_Read:BoundingBox
Lightprobe - ::Read_Probes:Lightprobe
Lightprobe - ::Reduce_Probes_1:Lightprobe
Image - ::Image_Read:Image
Lightprobe - ::Tweak_Probe_Color:ImageTypes
Lightprobe - ::Scale_Probes:ImageTypes
Lightprobe - ::Reduce_Probes_2:Lightprobe
Lightfield - ::Lightfield_Rasterize:Lightfield
Lightprobe - ::HDR_Video_Sequence:Lightprobe
Pointcloud - ::Structure_Fm_Motion:Pointcloud
Pointcloud - ::Pointcloud_Read1:Pointcloud
Pointcloud - ::Pointcloud_Read:Pointcloud
Pointcloud - ::Pointcloud_Read2:Pointcloud
Lightprobe - ::Pointcloud_Merge:Lightprobe
```

Figure 2: *The Depends user interface with the dependency graph visible on the left, the scenegraph in the lower-right, and the selected node's attributes in the upper-right.*

is used for lighting and look development in visual effects production. Katana uses a dependency-graph structure to clearly and efficiently bring assets together. It scales nicely with various techniques to cull and display data, and it's relatively simple to get working in existing visual effects pipelines. It is currently, however, focused on lighting and look development instead of the general execution of command-line programs. Depends is heavily inspired by Katana, but is tuned more to other aspects of production.

The scientific community has also developed workflow management software in recent years. Software packages such as Discovery Net [Rowe et al. 2003], the Taverna Workbench, and the Kepler Scientific Workflow System are all examples of how research scientists are able to use workflow management to create and share results on an individual level or for departments across different institutions. We found these programs to be inspirational, but quite tightly coupled with the fields they were developed in. Depends has been designed to incorporate ideas from the scientific workflow community, but adjusted to fit the unique challenges of visual effects production.

3 Depends

Depends is a workflow management software package [Figure 2] that organizes the execution of existing command-line programs into a directed, acyclic graph (DAG). Each command-line program is wrapped using a simple Python interface into a DAG node called a process node. The process node contains a series of typed inputs, typed outputs, and general attributes. An entire DAG of process nodes, their connections, and their attributes is henceforth referred to as a workflow. Depends executes workflows by writing execution scripts that run programs in the order they are described in the dependency graph.

A scenegraph, or collection of existing data at any point in the DAG, is tracked by Depends. This scenegraph is not as complex as in other workflow management software such as Katana, but is essential in understanding what data are available at each point in the dependency chain.

To maintain a smooth, consistent flow of information through the DAG, Depends allows the user to define collections of data that represent a single idea or asset. We call these collections of associated data, data packets. Data packets bind a set of related files together, making it easy for programs in the dependency graph to check and understand the information presented to them.

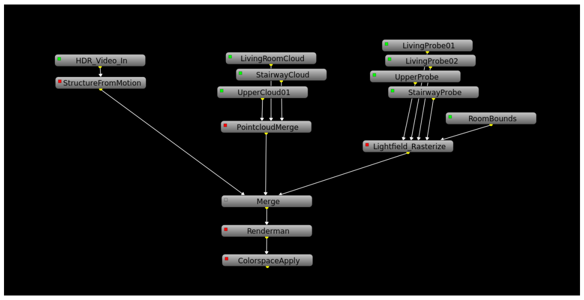


Figure 3: *An example of a directed acyclic graph in the Depends user interface. Process execution order, information flow, and data presence are visible at a quick glance.*

The data types and program DAG nodes in Depends are easily configured by a novice Python programmer, but nearly any artist familiar with a dependency graph interface is capable of using Depends to manage her workflow. A single programmer can therefore spend a short time configuring Depends, and those who are not familiar with programming can use it to organize their workflows.

3.1 The Directed Acyclic Graph

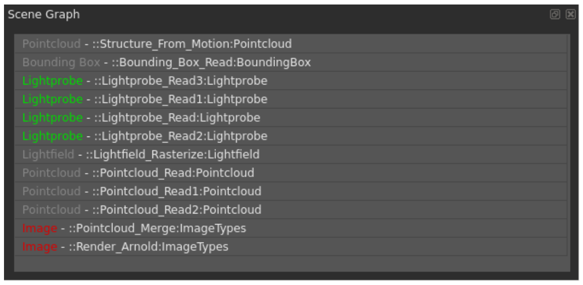
Depends uses a directed acyclic graph [Cormen et al. 2001] [Figure 3] to organize the execution of nearly any command-line software. The DAG structure clarifies the order in which programs will be executed. The DAG also helps the user to deduce which programs depend on other programs and which programs can run in parallel. The Depends graph is extensible, as it enables programs and their DAG nodes that didn't exist when the workflow was created to be incorporated at any time.

On a basic level, Depends tracks which nodes have already been run and no longer need execution. If all the data for all the process nodes exists on disk, and one of the parameters of a node has been changed, the DAG makes it clear which nodes depend on the new data and must be re-executed. This information is essential to managing complex workflows, and Depends features a number of user interface tools to quickly update attributes and re-run the workflow.

Further advantages of the Depends DAG user interface include the ability to select a process node and see which nodes it depends on, a list of which nodes can act as inputs to your node of choice, and if those nodes need to be executed or if their data are already present. Similarly, removing an operation in the middle of the DAG alerts the user to which operations may no longer have data available to them.

3.2 The Scenegraph

The Depends scenegraph [Figure 4] is a visual representation of what data are available at any location in the dependency graph. Scenegraphs are often associated with object transforms and the relationships between objects in a given space. Depends, however, presents the data as a flat hierarchy, showing the user exactly what is available and what can be ingested by a selected process node. This allows Depends to work with processes other than those that manipulate objects in three dimensional space. While this reduces the scenegraph's functionality in the context of lighting and look development, we have found it allows Depends to be more consistent and applicable in other contexts.


Scene Graph

Pointcloud - ::Structure_From_Motion:Pointcloud
Bounding Box - ::Bounding_Box_Read:BoundingBox
Lightprobe - ::Lightprobe_Read3:Lightprobe
Lightprobe - ::Lightprobe_Read1:Lightprobe
Lightprobe - ::Lightprobe_Read:Lightprobe
Lightprobe - ::Lightprobe_Read2:Lightprobe
Lightfield - ::Lightfield_Rasterize:Lightfield
Pointcloud - ::Pointcloud_Read:Pointcloud
Pointcloud - ::Pointcloud_Read1:Pointcloud
Pointcloud - ::Pointcloud_Read2:Pointcloud
Image - ::Pointcloud_Merge:ImageTypes
Image - ::Render_Arnold:ImageTypes

Figure 4: *An example of a scenegraph in the Depends user interface. Each entry represents data constructed by the DAG up to this point. The data packet type is written in green (ingestible by this node and present on disk), red (ingestible by the node and not present on disk), or grey (incompatible with this node). A unique string for the node name and output are also present.*

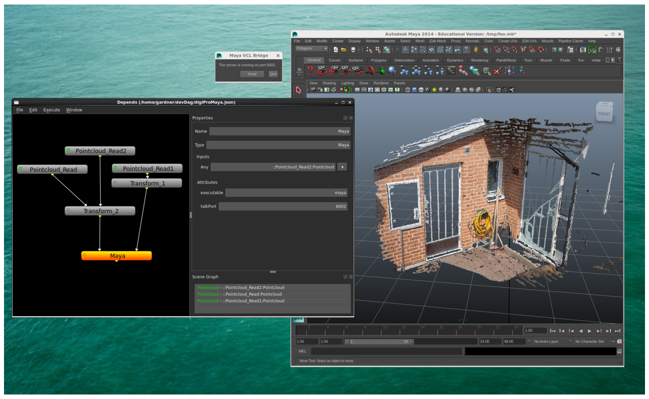


Figure 5: *The Depends user interface communicating with the commercial modeling package Maya to visualize and manipulate a point-cloud.*

3.3 Data Packets

A data packet is the name given to a collection of data that represents a single asset to be passed from process node to process node in the DAG. Bundling various files together simplifies the user experience of tracking data on disk. It also provides the Depends execution engine with the ability to type-check data flowing between process nodes, as nodes are defined with typed inputs and outputs.

Depends allows data packet definitions to inherit from other data packet definitions, letting process nodes ingest a variety of closely-related data as input. For example, a process node that filters an image would be designed to take an image data packet as input. A data packet representing a light probe image, which contains an image and a three-dimensional transform, should also be accepted by the image filter process node. Depends makes this possible by allowing the light probe data packet to inherit from the image data packet. This logical relationship is handled internally in the Depends architecture, simplifying workflow construction for the user.

Data packets are defined as a theoretical wrapper around general data on disk. These wrappers are constructed and used by Depends, but also may be useful outside the Depends user interface. To this end, the data packet definitions can be exported alongside the data they represent, and other software can refer to groups of data using the same organizational structures.

Finally, if a studio already has a method for grouping important information together, a single data packet type can be defined and all nodes can be configured to send and receive the type. File formats such as Alembic or GTO, are examples of ”bucket” files that may encompass the same functionality as a data packet definition. Wrapping all Alembic files with a single Depends data packet and passing those between process nodes is simple and effective.

4 User Interaction

Depends includes a number of other features that enhance its ability to organize an artist’s workflow, while maintaining a flexible and robust interface. Depends takes great care to ensure that each of these features are powerful, but as lightweight as possible, as we have found too many automated features limit an artist’s ability to do exactly what she wants.

One of the primary benefits of Depends workflow management is that the user does not write algorithms within the Depends infrastructure, but instead wraps existing software in operational units, or nodes, that can be organized into a DAG. Any software, commercial or homegrown, with a command-line interface can be incorporated into a Depends workflow. Shell scripts or collections of Python scripts are commonly used for this type of pipeline management in visual effects. The Depends user interface attempts to maintain the power of these scripts, but clarify complicated workflows.

Since Depends tracks data and does no data manipulation itself, it maintains a very small memory footprint. Each command-line application in a workflow is expected to manage its own memory. Depends, however, enables the user to configure portions of the DAG to run in parallel. A great deal of parallel activity may create resource conflicts. If simultaneous processes use too many resources on a host, the parallel assignments can be adjusted.

Depends contains a network socket layer that allows it to communicate with any external package that contains a similar interface. This permits the user to select a data element in the Depends scenegraph and quickly view and manipulate it in a 3d package such as Autodesk’s Maya [Figure 5]. Results can then be transferred back into the Depends workflow, allowing for, e.g., iterative improvement of coarse geometric alignment and other manual tasks.

Depends maintains a list of internal variables that can be referenced in any attribute field of any process node. A similar mechanism is used to reference environment variables from the shell Depends is run from, allowing workflows to be moved from place to place on a file system without needing to change dozens of individual parameters. Automated workflow wedging is also built in to Depends using this variable mechanism, as variables can assume numbers in a range, and portions of the workflow can be executed over these ranges.

Depends provides multiple options to preserve the history of its workflow files and the corresponding history of its results. First, it saves projects as plaintext JSON files that can be viewed and edited in an external text editor. These text-based workflow files can be tracked using a revision control package. The Depends user interface also offers a simple way to automatically increment the workflow version upon save, encouraging users to form habits that promote easy tracking of data provenance.

Executing entire workflows can be resource intensive. To avoid costly errors, Depends performs a series of checks before each time-consuming workflow execution. These include insuring all input files and output directories exist, and confirming all input and output types match. Hooks are available that run immediately before and after a process node’s primary execution function. These al-

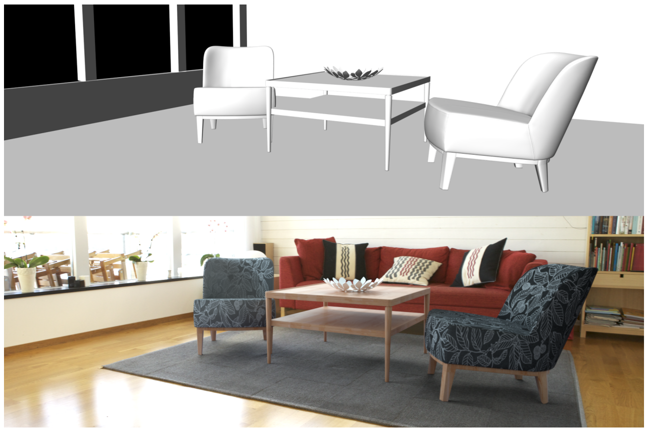


Figure 6: *A rendered image of synthetic furniture composited into a virtual photo set constructed from geometric scans, photographs, reflectance measurements, and high-dynamic-range video of a real-world environment.*

low for user-customizable sanity checks and tabulation of results for command-line programs that write an unknown number of files to disk.

Depends provides the ability to replace the default file browser with a browser custom-tailored to the way a studio accesses its data. This is useful for visual effects houses that store and retrieve data using asset management databases that require more than simple pathnames to refer to files, or various other types of custom file references.

Workflow execution can also be tailored to the needs of a studio. In its default configuration, Depends constructs execution sequences as shell scripts that can be run on a single machine. An experienced programmer, however, can write a plugin to send the execution information to a render farm manager, such as Pixar's Tractor or PipelineFX's Qube, to allow these specialized programs to manage the complex tasks of parallelization and load management.

Finally, Depends contains much of the user interface functionality artists are accustomed to. Undo and redo are robust and oriented towards losing as little work as possible. Documentation can be included for node attributes and quickly accessed by a user while constructing a workflow. File names in the user interface can have their versions incremented without having to enter text in multiple fields. And everything can be run from the command-line without having to open the UI, making it easy to create and execute a workflow's execution script without a graphics display.

5 Customization and Development

Depends can be used by artists with no programming experience or developers with Python skills. Using the program to construct workflows can be done by nearly anyone familiar with computer graphics user interfaces or dependency graphs. Creating node and data packet definitions, however, requires a rudimentary understanding of the Python programming language, and writing new execution engine backends or communicating with external programs are more fit for a developer experienced in these fields.

The freely-available Depends source code is developed to be as legible as possible. The Python codebase is relatively small and modular, and uses basic language functionality to do the majority of its tasks. The features of the program that are designed to be pluggable are separated from the rest of the source, and the entry points are designed to be clear. Depends ships with a handful of user interface commands to save development time, such as reloading all nodes from disk without having to restart the program and tracking exceptions in process nodes.

Figure 7: *Synthetic furniture rendered into a virtual environment reconstructed by the VPS project.*

Creating new data packet definitions is a simple matter of inheriting from a Python base class and adding a series of identifiers to a list owned by the base. Nodes that load these data packets off disk are automatically generated from the data packet definition classes at runtime, so no additional programming is needed. Process nodes require four functions to be overridden, three defining the input types, the output types, and the attribute names, and the fourth defining how the inputs and attributes are passed to the appropriate command-line program.

Data packet and node definitions are loaded off disk at startup using a shell environment variable that defines the location of each. This makes it simple to swap entire sets of data packet and process node definitions for different shows or shots that may be worked on concurrently.

Communication with external programs requires the developer to open a network socket in the external program and wait for commands to be issued from a process node in Depends. The format of the command varies from external program to program, but examples of each side of the communication mechanism for Autodesk's Maya are available with the source code download.

6 Example Use

We believe the core principles of Depends make it useful in a wide variety of situations. The following describes two of our own projects that Depends has improved and offers two additional examples in the context of visual effects that may also benefit from Depends.

Virtual photo sets (VPS) - We present a process we refer to as *the virtual photo set* [Unger et al. 2013a; Unger et al. 2013b]. The VPS toolchain has been developed in collaboration with IKEA Communications AB (the maker of the IKEA catalog), and is a completely data driven pipeline for measurement and reconstruction of real world lighting environments and material characteristics (reflectance, color, texture etc.), enabling robust, photo-realistic product visualization for catalogs and web applications with predictable results. This work is very similar to set reconstruction in the visual effects context.

An overview of the VPS pipeline implemented within Depends can be split into three steps:

1. *Scene capture* - A scene is captured using a set of input devices typically including: high resolution digital SLR cameras, HDR images and video, a laser scanner, and a spectrophotometer.

2. *Data processing and scene reconstruction* - The captured data is processed using computer vision algorithms as well as interactive methods, so an accurate model of the scene can be reconstructed with full artist control.

3. *Image synthesis* - The reconstructed environment consists of a geometric model textured with HDR-video frames and can be populated with virtual objects and directly used for *image based lighting* renderings.

A typical VPS user scenario is illustrated in Figure 1. The real world scene, a photo studio, is captured using: a Faro Focus 3D laser scanner, a sequence of raw images and 360° HDR panoramas from a digital SLR camera, a set of HDR-video sequences captured using a prototype multi-sensor HDR-video camera developed in our lab [Kronander et al. 2013b; Kronander et al. 2013a; Unger and Gustavson 2007; Unger et al. 2004], and a set of multi-spectral measurements of selected surfaces in the scene, captured using a PhotoResearch PR-650 Spectrascan.

Input nodes in the workflow load the data from the different capture sources. The merged laser-scan point clouds are decimated and tessellated, and in most cases are also manually adjusted. We use Maya through the network socket layer as the user interface for manual geometry adjustments. The SLR and HDR-video images are calibrated using process nodes for shading correction and HDR reconstruction, and then transformed to a common radiometric space using the spectrophotometer measurements. Using structure from motion (SfM) nodes [Sinha et al. 2009; Furukawa et al. 2009], and interactive alignment, the SLR and HDR images are aligned to the tessellated point-clouds. After camera calibration and registration, the image data is projected onto the recovered geometry.

The result is a reconstructed geometric proxy model representing the scene with re-projected HDR textures. The scene model can be populated with virtual objects and rendered using advanced image based lighting as shown in Figure 1 (right). Additional examples are shown in Figure 6 where an environment has been recovered and populated with virtual furniture (top) to produce photo-realistic renderings (bottom), and in Figure 7.

BRDF measurement and parameter fitting - In our second example, we have created a Depends workflow for capture of *Bi-Directional Reflectance Distribution Functions* (BRDFs). As illustrated in Figure 8, we use Depends nodes to: control the camera based capture hardware [Eilertsen et al. 2011], calibrate the input data, and fit BRDF model parameters such as Cook and Torrance [Cook and Torrance 1982], Ward [Ward 1992], Ashikmin and Shirley [Ashikhmin and Shirley 2000], and Löw et al. [Löw et al. 2012].

The capture hardware, see Figure 8 (top), is a gonioreflectometer with two arms, allowing a light source and a camera to move freely over the hemisphere above a material sample. The software that controls this device has been wrapped by a process node that presents the user with an interface for controlling the angular sample density and camera exposure. The BRDF is measured by illuminating the physical material sample from a set of angles over the hemisphere, and for each incident light angle measuring the outgoing radiance distribution. The captured images are processed by an image calibration node where shading correction, radiometric scaling, and other operations are performed. These results are then fed into a process node for numerical fitting of BRDF model parameters to the captured data. Finally, the fits can be inspected by comparing the graphs of the fitted BRDF models to the original data, by visualizing the lobes, or by rendering simple geometry as illustrated in Figure 8 (bottom).

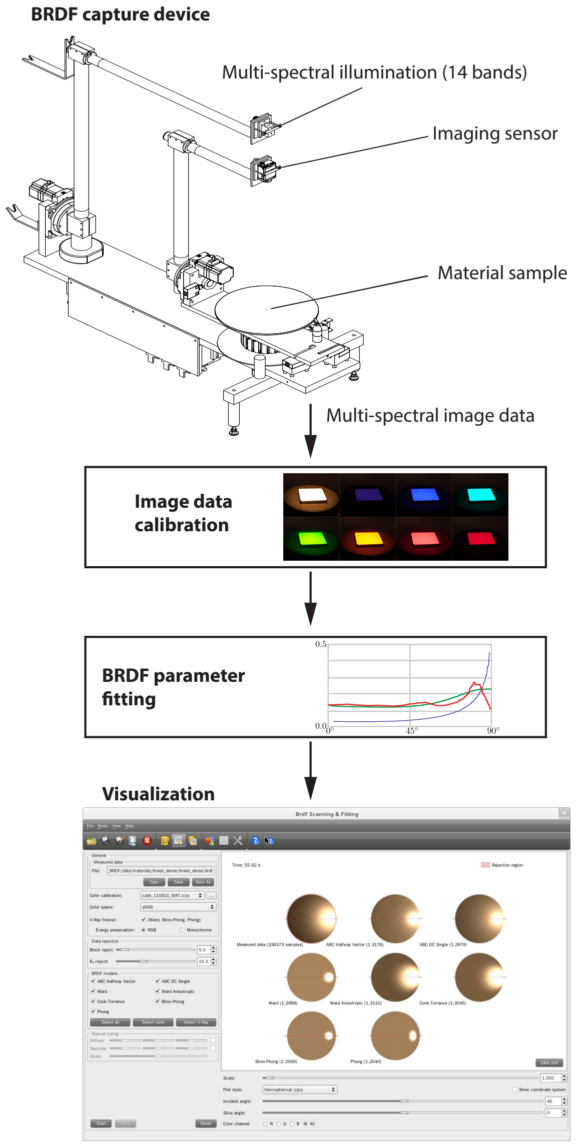


Figure 8: *The* Depends *workflow uses nodes to control our multi-spectral BRDF measurement hardware, process and calibrate the output image data, fit the parameters of BRDF models, and to visualize the results.*

Fluid surfacing - A potential use for Depends in a visual effects context is to assist artists in generating realistic surfaces of animated fluids. Fluid surfaces in visual effects often begin with animated particles, which are converted to level set representations, then transformed into polygons. One way to approach the transformation from particles to polygons is to create a large program with a complex user interface and a fixed data-transformation pipeline. This often generates great results, but may limit the flexibility of a talented fluid surfacing artist. The Depends methodology suggests the surfacing pipeline be split into multiple command-line programs that do very specific things. An artist can create a workflow in Depends then quickly reconfigure the workflow as new ideas or techniques come to light. Ideas can be quickly tested by rewiring the DAG to, for example, cause input particles to affect output geometry or filter level sets in multiple steps and combine the results to form a final volume. Managing a workflow using Depends gives the artist a great deal of control, as operational changes can be made without any input from a specialized tool programmer.

Camera calibration - Visual effects artists who calibrate cameras may also find Depends useful. Camera calibration often requires a collection of photographs of a fixed pattern. Tracking data for a single camera with a single lens is relatively straightforward, but once multiple cameras and multiple lenses are introduced, managing which calibration is associated with which lens, which camera body, and at which focus can be challenging. We've found Depends to work well for quickly recalling what information is associated with a given camera and the which inputs led us to derive that information.

7 Conclusion

Depends, the open source workflow management software, has been instrumental in reconstructing virtual photo sets from large collections of input data. In our experience, it has allowed us to organize data in a clear, consistent, and concise manner. It has also helped us revisit our projects later and quickly understand which inputs were used to generate our final renders. We have found it useful when sharing results with coworkers both in and outside our group, and believe the same results are possible in various visual effects contexts. We also hope releasing the source encourages additional development, helping Depends to grow in scope and utility.

Acknowledgements

We would like to thank our mentors and coworkers, Mike Root, Jim Hourihan, Scott Singer, and Jonathan Ramos for learning and coding with us while trying to make sense of the beautiful jumble that is visual effects production. Thanks also to Sony Pictures Imageworks and the Foundry for developing and releasing Katana, an inspiring piece of software that we anticipate will continue to grow, improve, and influence visual effects production well into the future. And finally, to the University of Linköping, for the support and belief that workflow management will benefit us all. This project was funded by the Swedish Foundation for Strategic Research (SSF) through grant IIS11-0081, Linköping University Center for Industrial Information Technology (CENIIT), and the Swedish Research Council through the Linnaeus Environment CADICS.

References

ASHIKHMIN, M., AND SHIRLEY, P. 2000. An anisotropic phong brdf model. *Journal of Graphics Tools 5*, 2, 25–32.

COOK, R. L., AND TORRANCE, K. E. 1982. A reflectance model for computer graphics. *ACM Transactions on Graphics 1*, 1, 7–24.

CORMEN, T. H., STEIN, C., RIVEST, R. L., AND LEISERSON, C. E. 2001. *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education.

EILERTSEN, G., LARSSON, P., AND UNGER, J. 2011. A versatile material reflectance measurement system for use in production. In *Proceedings of Sigrad2011*.

FURUKAWA, Y., CURLESS, B., SEITZ, S. M., AND SZELISKI, R. 2009. Reconstrucing building interiors from images. In *Proceedings of the IEEE International Conference on Computer Vision*, 80–87.

KRONANDER, J., GUSTAVSON, S., BONNET, G., AND UNGER, J. 2013. Unified HDR reconstruction from raw CFA data. In *Proceedings of the IEEE International Conference on Computational Photography*.

KRONANDER, J., GUSTAVSON, S., BONNET, G., YNNERMAN, A., AND UNGER, J. 2013. A unified framework for multi-sensor HDR-video reconstruction. *Accepted for publication in Signal Processing: Image Communications*.

LÖW, J., KRONANDER, J., YNNERMAN, A., AND UNGER, J. 2012. BRDF models for accurate and efficient rendering of glossy surfaces. *ACM Transaction on Graphics 31*, 1 (January), 9:1– 9:14.

RAYMOND, E. S. 2003. *The Art of UNIX Programming*. Pearson Education.

ROWE, A., KALAITZOPOULOS, D., OSMOND, M., GHANEM, M., AND GUO, Y. 2003. The discovery net system for high throughput bioinformatics. *Bioinformatics 19*, suppl 1, 225–231.

SINHA, S. N., STEEDLY, D., AND SZELISKI, R. 2009. Piecewise planar stereo for image-based rendering. In *Computer Vision, 2009 IEEE 12th International Conference on*, 1881 –1888.

UNGER, J., AND GUSTAVSON, S. 2007. High-dynamic-range video for photometric measurement of illumination. In *Proceedings of Sensors, Cameras and Systems for Scientific/Industrial Applications X, IS&T/SPIE 19th International Symposium on Electronic Imaging*, vol. 6501.

UNGER, J., GUSTAVSON, S., OLLILA, M., AND JOHANNESSON, M. 2004. A real time light probe. In *In Proceedings of the 25th Eurographics Annual Conference*, vol. Short Papers and Interactive Demos, 17–21.

UNGER, J., KRONANDER, J., LARSSON, P., GUSTAVSON, S., LW, J., AND YNNERMAN, A. 2013. Spatially varying image based lighting using HDR-video. *Computers & graphics 37*, 7, 923–934.

UNGER, J., KRONANDER, J., LARSSON, P., GUSTAVSON, S., AND YNNERMAN, A. 2013. Temporally and spatially varying image based lighting using HDR-video. In *Proceedings of EUSIPCO '13: Special Session on HDR-video*.

WARD, G. J. 1992. Measuring and modeling anisotropic reflection. *SIGGRAPH '92 26*, 2, 265–272.