DIRSIG: Image and Data Simulation

Scott Brown, Adam Goodenough and Rolando Raqueno
Modeling and Simulation Group
Digital Imaging and Remote Sensing Laboratory
Rochester Institute of Technology
Rochester, NY
DIRSIG: What is it?

- RIT has been developing a 3D image and data (non-image) simulation environment for 20+ years called the “Digital Imaging and Remote Sensing Image Generation” (DIRSIG) model.
- It is used by the civil, commercial and military community to model what complex remote sensing sensors would collect.
- DIRSIG is a ray-tracing based renderer that is specialized for modeling image data for ground, airborne and space-based remote sensing systems.
  - You make a virtual world using 3D geometry and then you probe that world with rays to form images, find reflections, etc.
- DIRSIG is used by engineers and scientists to model very specific imaging and non-imaging sensors.
  - DIRSIG is more like a hardware simulator than a common renderer.
  - You don’t turn a “knob” until it “looks right”. You input a specific numerical value that has engineering importance.
DIRSIG: The Basics

Modeling a Real-world Target

Geometry:
3D geometry model by “natman” (Neil Taylor)
Obtained from BlendSwap CC-BY License

Materials:
Surface (Bulk) Optical Properties
Thermodynamic Properties

Object Database:
Different objects
Geometry variants
Material variants

Material Attribution

Platform Motion:
Dynamic location/orientation

Atmosphere:
Radiative Transfer
Weather (Thermodynamics)

Platform:
Attach multiple instruments

Collection Data:
Date and Time

Data Products:
Low-Level Imagery
Platform Ephemeris
Truth Data

Instruments:
Imaging and Non-Imaging Instruments

Scene Assembly

DIRSIG5 Phase #2 Kickoff 20 Oct 2015
DIRSIG: Image and Data Simulation

- An image and data simulation tool for modeling remote sensing systems:
  - Pan and RGB
  - Multi/Hyper-spectral
  - Thermal
  - Video
  - Polarization
  - Low light/night
  - Plumes/Water/Clouds
  - LIDAR/LADAR
  - SSA*
  - SAR*

* Experimental application areas
DIRSIG: State of the Model

• Software and Updates
  – Binary software releases on Windows, Linux and Mac
    • Software is “closed source” to avoid private forks that would dilute validation and verification. This approach has been encouraged by our government partners.
  – Quarterly software updates to all registered users (free of charge, internally funded)

• User Base
  – Currently 450+ registered users, with 3-5 training course sessions that are held around the country and produce 40-75 new users annually.
  – Users install on laptops, desktop and smaller servers. As of the 4.5.4 release (Summer 2014) Windows accounts for nearly 75% of all downloads.

• Usage
  – Heaviest usage in general EO/IR and LADAR/LIDAR
  – Biggest growth over past 2-3 years in LADAR/LIDAR and ABI
  – Hot new interest areas are SSA and SAR
DIRSIG5

Creating the next generation, remote sensing centric image and data simulation system to power system design trade and data exploitation studies
DIRSIG5: What to expect

• Improved scalability
  – Automatically uses multi-threading on multi-core laptops/desktops.
  – Extendable through MPI to execute on HPC clusters.

• Improved baseline radiometry
  – Harder for users to produce poor results (reduced technical risk).

• Faster execution when it matters
  – Significantly faster start-up time and computations for non-trivial radiometry.

• Lowered barrier to entry
  – Much less complex and easier to get new users up to speed.

• Reduced code complexity
  – Decreases risk of bugs and simplifies long term development efforts.
DIRSIG5: Path Tracing Radiometry

- DIRSIG5 will migrate to using “path tracing” to compute the flux onto pixels.
  - Each ray generates a single path or series of scattering events leading to a source.
  - Importance sampling focuses rays on important paths:
    \[
    \int_{A} f(a)da = \int_{A} f(a) \frac{p(a)}{p(a)} da = E \left\{ \frac{f(a)}{p(a)} \right\}.
    \]
  - The average converges to the correct solution.
- Path tracing is efficient and can be easily parallelized.

*many sensor rays generate similar paths for relatively constant computation time across threads*
DIRSIG5 Implementation Plan

• Spiral Development Approach
  – Initial phase focused on fundamental code development
    • Major release by end of March 2016 with limited functionality
    – Followed by series of parallel, spiral development activities

• Software Release Schedule
  – After the initial release, there will be releases approximately every 2 months
  – Multiple releases allows for feedback from user community during development

• Testing Plan
  – Development of rigorous and continuously utilized testing to verify and validate consistent system performance during the entire development effort.
DIRSIG5 Implementation Schedule

2015 | 2016 | 2017
---|---|---

**DIRSIG5 Code Development**
- High Performance Core
- Modality Support (PAN, MSI, HSI)
- Scene Support
- Atmosphere Support
- Sensor Support
- Deployment and Testing

**DIRSIG5 Spiral Enhancements**
- Milestone #1 (March 31, 2016)
- DIRSIG5 Spiral Releases (approx. every 8 weeks)

**Spiral Enhancements**
- Parallelization Spiral
- GPU Acceleration Spiral
- Scene Geometry Spiral
- Scene Optical Property Spiral
- User-defined Source Spiral
- Sensor API Spiral
- Atmospherics Spiral
- Temperature Spiral
- Testing Spiral
- User Experience Spiral

**Future Capabilities**
- LIDAR Spiral
- SSA Spiral
- Polarization Spiral
- Radar/SAR Spiral

**Project End:**
August 31, 2017

DIRSIG5 Phase #2 Kickoff: 10 October 2015
Quarterly Status Meetings: 20 October 2015
DIRSIG5 Spiral Highlights

• High Performance Core
  – CPU-accelerated core (Intel Embree based) and GPU-accelerated core (NVIDIA OptiX based)
  – Multi-threaded support (span cores on a single machine) and MPI support (span many machines in a cluster)
  – Current computing platforms (64-bit Windows, Mac OSX and Linux)

• Sensor Support
  – Introduction of public sensor API (later)

• Atmospherics
  – MODTRAN6 integration, 6S integration, direct adjacency modeling, direct turbulence modeling

• User-Defined Sources (area sources, direct viewing)

• Testing and Validation
  – More automated release QA, continuous integration, etc.
# Program Risk Assessment

<table>
<thead>
<tr>
<th>Risk Category</th>
<th>Evaluation Criteria</th>
<th>Evaluation Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Requirements</td>
<td>Are the requirements fully established and stable?</td>
<td>Architecture Study established requirements.</td>
</tr>
<tr>
<td>Software Development</td>
<td>Is the development process supported by a compatible set of procedures, methods, and tools?</td>
<td>Common software architecture defined and development tool chain is well established.</td>
</tr>
<tr>
<td>Testing</td>
<td>Is there an established approach the code during all development spirals?</td>
<td>The testing environment has been established, but the detailed procedures, methods, and tools will be developed in the first 6 months of the project.</td>
</tr>
<tr>
<td>Schedule</td>
<td>What are the largest schedule risks and how are they being mitigated?</td>
<td>Limited schedule risk, but with some staffing holes to be filled.</td>
</tr>
<tr>
<td>Budget</td>
<td>Is the budget sufficient and how will it be tracked?</td>
<td>The budget fully funds the development team that has been defined to execute a successful project.</td>
</tr>
<tr>
<td>Staffing</td>
<td>Is the staff inexperienced or understaffed?</td>
<td>3 of the 4 team members have over a decade of experience working together on modeling and simulation as well as on the recent DIRSIG5 architecture study. There is one hire required to support the software development activities.</td>
</tr>
<tr>
<td>Management Environment</td>
<td>How will inter-team communication and expectations be accomplished?</td>
<td>The RIT development team is not co-located requiring additional effort to communicate. However, a number of proven techniques are being employed to insure optimum communication and team dynamics.</td>
</tr>
</tbody>
</table>

**Low Risk:** risks understood and well positioned to address

**Moderate Risk:** risks understood, but may require additional effort to address

**High Risk:** risks may not be understood or not well positioned to address
Technical Requirements

• Scope
  – Architecture and prototyping study (completed in Phase 1)
  – Solicit priorities from user community (completed in Phase 1)
  – Define the scope of the initial release (completed in Phase 1)
  – Create detailed scope for each enhancement spiral (by Jan 2016)

• Schedules
  – Create detailed schedule for initial release (by Nov 2015)
  – Create detailed schedule for each enhancement spiral (by Jan 2016)
Software Development

• Development team is distributed but uses centralized resources
  – Shared computing resources at RIT
  – Using the “Slack” messaging platform as our communications tool for all aspects of the project.

• Key software technologies identified and in place
  – Compilers (GCC, Microsoft and Intel) and build tools
  – Revision control (Subversion)
  – Continuous integration (Buildbot)
  – Problem and bug tracking (Bugzilla)
  – 3rd Party APIs/Frameworks (Embree, OptiX, Qt, GoogleTest, etc.)
Testing

• Significant focus on developing API, plugin, integration and end-to-end testing that will be integrated into a Continuous Integration (CI) environment.
  – Continuous Integration is a process where software is monitored and automated, periodic software compilation and testing is performed.

• Internal user testing at RIT
  – We will migrate all faculty, staff and students at RIT to the new model for general use as soon as possible.

• External user testing
  – We will start a beta tester program to have external users run the code.
Schedule

• Well established plan for the 1\textsuperscript{st} deliverable (initial limited capability release) by the end of March 2016 (5 months)
  – This is one of the most important milestones for the entire project
• Additional detail required for Spiral Development modules
  – Want to initiate Fundamental Code development
  – Will present more detailed plan/schedule at next status meeting (Jan 2016)
• In addition to constant interaction, weekly team meetings to check status and identify issues
• Customer interaction at status update meetings
  – Open to other mechanisms (monthly telecons?)
Budget

• Budget in good shape
  – Project is adequately funded for success

• Budget Tracking
  – RIT accounting system provides monthly expenditure reports by categories (labor, travel, material, indirects, etc.)
  – Actuals are compared to planned spending and any variances are analyzed
  – Custom budget tracking software allows for routine cost to completes and variance repairs
Staffing: The RIT Team

- Dr. Scott Brown – Principal Scientist [80% - 90% effort]
  - 22 years @ RIT
  - Technical and programmatic lead
  - Higher-level code development

- Dr. Adam Goodenough – Senior Scientist  [80% - 90% effort]
  - 8 years @ RIT (+8 years as student)
  - Chief Software Architect
  - Lower-level code development

- Dr. Rolando Raqueno – Senior Scientist  [90% effort]
  - 25 years @ RIT
  - Documentation and system software testing

- TBD Software Support [100% effort]
  - GUI development, code development and testing

- Michael Richardson – Project Manager  [15% effort]
  - 30 years industry experience, 15 years @ RIT
  - Customer reporting, financial management
Staffing: Additions

• Areas of expertise needed
  – High performance computing, massive parallel environments
  – User and Data Interfaces

• Possible solutions
  – Experienced student (6-month co op)
  – External consultants (for example, colleagues at UT Knoxville)

• Job descriptions complete and being prepared for release
  – Need to hire or contract software support by Jan 2016
Management Environment

• The development team is spread out across 3 states (NY, MA and VT), but this team has been virtually working together for 8+ years.
  – Constant interaction throughout the day via messaging and video conferencing.
  – Frequent face-to-face time in Rochester and at DIRSIG training sessions.

• We have migrated to “Slack” as our primary messaging and general communication platform.
  – “Slack” is a commercial messaging/communication platform that has become very popular with software development teams.
  – “Slack” has an open API that allows messages generated by software tools to be integrated.

• Mike Richardson will lead contract programmatic and Scott Brown will lead technical project management.
Technical Details
Initial Development Sprint

A 6-month development sprint to get a minimal, but non-trivial model implemented.
Initial High Performance Core

• CPU Accelerated Framework
  – CPU-accelerated ray tracing engine (Intel Embree) coupled with uni-directional path tracing (PT)
  – CPU-accelerated spectral vector calculations (ported from DIRSIG4)

• Parallelization
  – Multi-threaded support (span cores on a single machine)
  – No initial MPI support (span many machines in a cluster)

• Currently supported computing platforms
  – 64-bit Windows, Mac OSX and Linux
Initial Modality Support

- Passive PAN, MSI and HSI EO/IR
  - Includes initial reflected low-light
- Limited thermal and SSA support
  - External temperature solutions only
  - Primitive “earth as a background”
- No active EO (LADAR and LIDAR)
- No polarization
- No active RF (RAR and SAR)
Initial Scene Data Model

- Using high performance HDF scene data model with conversion tools from existing DIRSIG4 scene models
- Polygon based geometry (GDB and OBJ files) but not primitives (spheres, cubes, cones, etc.)
  - Static instances (houses, trees, etc.)
  - Affine dynamic (moving cars) but not deformable geometry (wave models)
- Materials will be spectral, but limited BRDF support
  - 100% diffuse, 100% specular or Ward (specular + diffuse)
  - Most advanced DIRSIG4 BRDF models not supported (yet)
- Limited temperatures
  - External (MuSES, NGIC IFS, etc.) or maps (no built-in temperature model yet)
- User-defined point sources (illumination only)
  - No area sources, no direct viewing
- Limited material related map support
  - Material, mixture, texture and temperature
- No geometry related map support
  - Bump, normal, displacement, decal, etc.
- No volumes (clouds, plumes, water)
Initial Atmospherics

- Radiometry
  - Analytical model (user supplied spectral direct and diffuse model)
  - Initial MODTRAN-driven model (improved ADB type approach)
- Exo-Source ephemeris models
  - User-defined (static) and DE-405 prediction
Initial Sensor Modeling

• Basic camera geometry model and spectral response support
  – Conversion tools to ingest DIRSIG4 parameterized focal plane geometries
  – No data-driven focal plane model (yet)

• Data-driven platform motion
  – Location/orientation vs. time
  – No algorithmic motion models (for example, the SGP4 propagator) (yet)

• Initial set of raster truth collectors

• No public sensor API (yet)
Initial Scheduler, Reporting and Testing

• Minimal work scheduler and dispatcher
  – Simple, brute force scheduling of pixels vs. time.

• New logging and reporting system
  – Required for parallel execution where many threads of code are simultaneously executing.
  – Flexible level-of-detail reporting to help with debugging and performance evaluation.

• Initial testing environment
  – Basic unit, integration, functional, smoke and regression testing system in place.
  – All tests will be incorporated into a continuous integration system at RIT to provide automated “around the clock” compiling and testing.
Initial Capability Summary

• This initial sprint would result in a new foundation for the next 10 years of DIRSIG.
  – We believe we have identified an approach and architecture that will allow the model to adapt and tackle remote sensing modeling challenges for many years to come.

• This will be a non-trivial tool that would appeal to users enough to test it and provide needed feedback on deployment issues and performance experience.

• Application areas
  – Conventional airborne and space-based remote sensing users will have a nearly fully functional tool at this point.
  – But some specific (and popular) modalities will not be available at this point.
Enhancement Spirals

Ongoing efforts to add features to various areas of the model to recreate the full functionality of DIRSIG4.
Enhanced Parallelization Spiral

- Message Passing Interface (MPI) implementation
  - Span work across a cluster of computers
- Improved scheduler and dispatcher
  - Addresses smarter time vs bands vs pixels scheduling
- Deployment testing and tuning at RIT
  - Various on-campus clusters available
- Deployment testing and tuning in field (external users)
- Documentation of deployment requirements and setup

```bash
mpirun -np 4 ./myapp <args>
```
GPU Acceleration Spiral

• We believe we can squeeze as much as 10x better performance out of a single workstation class machine outfitted with a high-end GPGPU card.
  – For example, the $2000 NVIDIA K80 with 24 GB of memory
• GPU-accelerated core implementation utilizing NVIDIA’s OptiX ray tracing framework
  – Minimal prototype developed during Phase #1
• Deployment testing at RIT
  – Various machines and clusters available
• Deployment testing on 3rd party machines
  – Rented access
• Deployment testing in field (external users)
• Documentation of deployment requirements and setup
**Scene Geometry Spiral**

- Add API support for deformable dynamic geometry and provide documented wave model source code as an example.
- Add support for destructive (replacement) dynamic geometry
- Add support for geometry related maps and modifiers
  - Bump maps, normal maps, displacement maps
  - Evaluate decal maps vs. external cutting and fitting
- Address primitive geometry needs (ideal spheres, cubes, cones, etc.)
- Address volume geometry needs (clouds, plumes, water, etc.) including potential use of OpenVDB
Scene Optical Property Spiral

- Complete icosahedron BSDF implementation
  - Add support to scene compilation workflow to fit all existing BRDF models to data model.
- Evaluate advanced path tracing algorithms (BPT, UPDP, etc.) for volume scattering applications
User-Defined Source Spiral

• Provide direct viewing of sources
  – Add area sources (“JumboTron” type sources)
  – Add source assembly workflow (point sources combined with reflectors) to directly view lights.

• Add bi-directional path tracing (BPT) to core LTA support to improve performance

Times Square from the air
(GeorgeEn @ GlobalImages.net)
Sensor API Spiral

- Formalized public API for embedding DIRSIG5 radiometry engine within 3rd party sensor modeling workflows
- Work with 3rd party users to design and develop API
- Implement and document API
- Develop basic model at RIT with documented source code available as example
  - Provides users with basic toolbox to produce image products in digital counts.

Rolling shutter demo (Goodenough @ RIT)

Landsat Level-0 type data @ RIT

Pushbroom striping @ RIT
Atmospherics Spiral

• Improved surface weather data model
  – Helps with integration with 3rd party thermal models
• Completion of backend façade API and implementation
  – MODTRAN6 support for line-by-line calculations required for thermal IR spectroscopy
  – Add 6S support (provides polarized atmosphere option)
• Explore spatially varying atmosphere approaches
  – RIT based MODTRAN extraction approach vs. getting SSI to open up MCScene (3D MODTRAN)
• Turbulence evaluation and approach
  – Can we directly incorporate turbulence into the path tracing? Potential MS level project topic*
• Adjacency evaluation and approach
  – Can we directly incorporate turbulence into the path tracing? Potential MS level project topic*
Temperature Spiral

- Need built-in temperature solution model
  - Path tracing will need temperatures on first, second, etc. bounces
  - THERM is very slow and requires a lot of ray tracing to compute solar insolation histories

- Approaches
  - Facet-scale temperatures produce limited fidelity and place constraints/requirements on facet scale or resolution.
  - Phase #1 prototyped a new THERM-inspired solution that would compute sparse temperatures at points in the scene.

- Other options
  - Access to Scalable Thermal Analysis for Real-time Signatures (STAR) and it is a multi-physics CFD has been offered by project lead.
User Experience Spiral

• Improvement and porting of select DIRSIG4 graphical user interfaces (GUIs) to release with DIRSIG5.
  – For example, improved interface to the current Platform Editor (sensor payload).

• Development of new graphical user interfaces (GUIs) to release with DIRSIG5
  – For example, a GUI interfaces to control and/or interact with the BRDF data model fitter.

• Development of user documentation

• Exemplar scene production and methodology documentation using 3D content creation experts.
Status Update

Significant progress already made
Software Architecture

Formalization of the Radiometry Engine
Core Geometry Engine

Path Generator
Private object with abstract interfaces

Path Processor
Private object with abstract interfaces

Radiance Interface
Fundamental Public API

Resource Scheduler
Private Resource Control Object

D5 Core
State Changes (spectral, temporal)

Important: this diagram starts at the radiance interface level – no platform/sensor information is included (or is relevant)

Plug-in Models
Geometry Models
Dynamic Motion Models
Optical Property Models
Atmosphere Models
Medium Models
Temperature Models
Whole Earth Models
Version/Validation Control and Transparency

Individual model interfaces
Scene Component Definitions
Atmosphere Definitions
Whole Earth Definitions

Internal/Public Plug-in API
Automated Testing with DIRSIG4

• Over the summer (2015) we prototyped a Continuous Integration (CI) environment using the DIRSIG4 code base
  – Continuous Integration is a process where software is monitored and automated, periodic software compilation and testing is performed.

• Current environment uses Buildbot (open source)
  – We have Buildbot slaves running on a Linux and Mac system (Windows coming soon).
  – Buildbot automatically compiles and/or runs tests on both slaves:
    • When any change is made to the source code, the code is compiled.
    • Every night the current code is compiled and the unit test suite (400+ test sets, 1200+ individual tests) run.
    • Every Friday night the current code is compiled and every demo scenario is run (there are currently 85 demo scenarios) and the output for a select set (15 at this time) is checked against established metrics.
Messaging/Communication with “Slack”

• It is possible to integrate software generated messages into “Slack”. In our case, we have integrated key development tools into our “Slack” configuration:
  – Subversion (SVN), for software revision control.
  – Buildbot, for continuous integration.
  – Bugzilla, for bug tracking (planned).

• This provides the development team a single platform for individual communication, group communication, observing code updates and monitoring software integrity.
Any code updates are automatically communicated to the team via a dedicated channel in Slack.
Buildbot sees the source code change.

Buildbot then triggers compiles on the slaves to verify code integrity.

The Buildbot web console lets us monitor what the system is doing.
Buildbot posts the results of all builds and tests to a dedicated channel in Slack.

Buildbot alerted the team that the weekly demo verification tests were failing on Mac OSX but not Linux.
The Buildbot web console lets us browse recent compilation and test tasks.

![Buildbot Console Screenshot](http://brownbox.cis.rit.edu:8010/one_line_per_build)

<table>
<thead>
<tr>
<th>Time</th>
<th>Revision</th>
<th>Result</th>
<th>Builder</th>
<th>Build #</th>
<th>Reason</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 13 15:08</td>
<td>16189</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#116</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 15:08</td>
<td>16189</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#116</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 13:28</td>
<td>16188</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#115</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 13:28</td>
<td>16188</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#115</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 11:28</td>
<td>16187</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#114</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 11:28</td>
<td>16187</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#114</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 04:00</td>
<td>16186</td>
<td>SUCCESS</td>
<td>MacOsXUnitTest</td>
<td>#40</td>
<td>The Nightly scheduler named 'NightlyUnitTest' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 04:00</td>
<td>16186</td>
<td>SUCCESS</td>
<td>LinuxUnitTest</td>
<td>#40</td>
<td>The Nightly scheduler named 'NightlyUnitTest' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 03:00</td>
<td>16186</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#113</td>
<td>The SingleBranchScheduler scheduler named 'NightlyTrunkCompile' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 13 03:00</td>
<td>16186</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#113</td>
<td>The SingleBranchScheduler scheduler named 'NightlyTrunkCompile' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 13:48</td>
<td>16186</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#112</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 13:48</td>
<td>16186</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#112</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 11:28</td>
<td>16185</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#111</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 11:28</td>
<td>16185</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#111</td>
<td>The SingleBranchScheduler scheduler named 'Dirsig4TrunkChange' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 04:00</td>
<td>16184</td>
<td>SUCCESS</td>
<td>MacOsXUnitTest</td>
<td>#39</td>
<td>The Nightly scheduler named 'NightlyUnitTest' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 04:00</td>
<td>16184</td>
<td>SUCCESS</td>
<td>LinuxUnitTest</td>
<td>#39</td>
<td>The Nightly scheduler named 'NightlyUnitTest' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 03:00</td>
<td>16184</td>
<td>SUCCESS</td>
<td>MacOsXTrunkCompile</td>
<td>#110</td>
<td>The SingleBranchScheduler scheduler named 'NightlyTrunkCompile' triggered this build</td>
<td>Build successful</td>
</tr>
<tr>
<td>Oct 12 03:00</td>
<td>16184</td>
<td>SUCCESS</td>
<td>LinuxTrunkCompile</td>
<td>#110</td>
<td>The SingleBranchScheduler scheduler named 'NightlyTrunkCompile' triggered this build</td>
<td>Build successful</td>
</tr>
</tbody>
</table>
Testing Plan for DIRSIG5

• We plan to use the Buildbot CI system to perform automated testing with DIRSIG5.
  – The fledgling DIRSIG5 Core code is already being monitored and tested by Buildbot.

• What to test?
  – The APIs that form the interfaces between each component of the code have to be robust and respond appropriately to both valid and invalid data.
  – Plugins that implement a feature via an API need to be tested under a variety of conditions to verify they perform their task correctly.
  – Eventually full simulations will be run and the output checked.
    • Simulations submitted with bug reports can be archived and retested (regression testing)
DIRSIG5 Testing: Current Status

- The development of the CPU-accelerated Core Geometry Engine (using Embree) is nearly complete.
  - The Core Geometry Engine API (shared by the CPU and GPU accelerated implementations) is well established.
  - The creation of unit test suites for this API (using Google’s open “GoogleTest” framework) is well underway.

- Parallel API and Test Development
  - The API architect (Adam) designs, documents and implements the API.
  - The Tester (Rolando) creates tests based on the documentation to confirm that the API behaves as expected.
    - It must correctly respond to valid and invalid input.
  - Although Rolando can look at Adam’s underlying code, the point is to have the tester treat the component under test as a black box.
The current DIRSIG5 Core is automatically compiled and tested by Buildbot after any code change.

Output of the current DIRSIG5 Core API unit tests viewed via the Buildbot web console.
Summary

• The phase 1 effort laid the foundation for this implementation phase.
  – The software architecture has been defined and the enabling technologies have been identified.
  – The core team is in place and the project is underway

• Original schedule from May 2015 has been adjusted to reflect new start date

• To Do Items:
  – Schedule next quarterly (in person) meeting (Jan 2016)
  – Schedule weekly or monthly status update meetings (telecon?)