Flash LIDAR Emulator for HIL Simulation

Paul Brewster
NASA Langley Research Center
ModSim World Conference
October 14th, 2010
Autonomous Landing and Hazard Avoidance Technology

• Introduction

• Problem

• Emulator Development

• Application

• Results

• Future Work
Introduction

• Autonomous Landing and Hazard Avoiding Technology (ALHAT/ETDPO)

• **Goal:** Develop and deliver a TRL 6 lunar GNC descent and **landing subsystem to place humans and cargo safely, precisely, repeatedly and autonomously anywhere on the lunar surface under any lighting conditions within 10’s of meters of certified landing sites**

• **Approach:** During the Approach phase, use three LIDAR systems to automatically scan the landing site, detect safe landing areas, and navigate to a determined safe area
Organization

- NASA Johnson Space Center
  - Program Management
  - Hardware-in-the-Loop Testing (HAST)
  - Avionics (APB)
- NASA Langley Research Center
  - LIDAR Sensors
  - 6DOF Simulation (POST2)
- NASA Jet Propulsion Laboratory
  - Hazard Detection Algorithms (TSAR)
  - System Integration
- Draper Labs
  - GNC algorithms
  - Navigation Filter
- Applied Physics Laboratory
  - Lunar Science
  - Lunar Terrain Models
Autonomous Landing and Hazard Avoidance (ALHAT)

System Block Diagram

- **Navigation & Guidance System**
  - Commands, NAV State
  - Safe Sites

- **Avionics Processing Box**
  - Velocity Measurements
  - Altitude Measurements

- **TSAR Processor**
  - Op Cmds
  - Image frames/Maps

- **Flash LIDAR System**
  - Op Cmds

- **Doppler LIDAR Velocimeter**
  - Op Cmds

- **LIDAR Altimeter**
  - Op Cmds
• Flash LIDAR
  – Fires a laser pulse, measuring the time for the pulse to return back to the camera, calculating the distance
  – Uses an array of sensors to create an image of distances, rather than a single point

• Doppler LIDAR Velocimeter
  – Fires three lasers in orthogonal directions
  – Determines velocity by measuring the doppler shift of the return beam

• LIDAR Altimeter
  – Fires a single laser pulse, measuring the time of return
  – Calculates the distance using a single point
Problem

- **Problem:** How do we develop, test, and evaluate the ALHAT system in a lab environment?
  - System components are being developed in four independent organizations
  - Impractical to use real LIDAR in a closed loop, hardware-in-the-loop, real-time lab environment
  - Physical constraints
  - Schedule constraints
  - Cost constraints

- **Solution:** Use a functionally equivalent **software emulator** to replace the LIDAR systems
Emulator Requirements

- Complies with Flash LIDAR Interface
  - Input
    - Command & Control
  - Output
    - 256 x 256 Range Image
    - 256 x 256 Intensity Image
    - 30 Images/Second

- Identical Hardware Interfaces
  - CameraLink (Images)
  - RS-232 (Command & Control)

- Similar Image Quality
  - Noise/Signal Ratio
  - Dead Pixels
  - Precision

- Integrates into HAST framework
  - Input
    - Sensor position & orientation (Ethernet)
    - Lunar Terrain Data (Pre-computed) (5000 x 5000 DEM)
Emulator Interfaces

- Terrain DEM
- Simulator Data
- Command/Control

Flash Lidar Emulator

- Pre-computed
- TCP/IP
- CameraLink RS-232
- CameraLink

Range/Intensity Maps
Emulator Block Diagram

Autonomous Landing and Hazard Avoidance (ALHAT)

Simulation Environment

TCP/IP

HAST ICD

Channel Link Interface

RS-232 Command & Control

Time Signal

Flash LIDAR Sensor Emulator

Future Work

DemMaker Terrain

NASA PDS

Range/Intensity Calculation

Sensor Model

Signal Processing Interface

Digital Signal Processing

Sensor Controller
Range/Intensity Calculation

- Create a triangle mesh from the DEM data (5000*5000*4 triangles)
- For each pixel on the focal image plane, create a ray from the camera position through the pixel (256*256 rays)
- Range is the distance from the camera position to the point where the ray intersects the triangulated terrain
- Intensity is \( \text{reflection} \times \cos(\text{incidence\_angle}) \) at that pixel
Range/Intensity Optimizations

**Problem:** The non-real-time implementation of the Flash LIDAR takes several seconds per frame. How do I implement the emulator for real time?

- Test intersection of 65,536 rays with 100,000,000 triangles, 30 times a second

**Solution:** Use optimization techniques from several computer fields:

- Computational Geometry
- Ray-Tracing
- Parallel Processing
- Vector CPU processing
- General-Purpose computation on Graphics Processing Units
### Computational Geometry

#### Autonomous Landing and Hazard Avoidance (ALHAT)

<table>
<thead>
<tr>
<th>Un-partitioned</th>
<th>Quad Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of triangles</td>
<td>Terrain recursively subdivided into 4 partitions forming a 4-way hierarchical tree</td>
</tr>
<tr>
<td>Each ray is tested against each triangle</td>
<td>Each ray is tested against the parent partition</td>
</tr>
<tr>
<td></td>
<td>If intersected, the ray is tested against the child partitions</td>
</tr>
<tr>
<td>O(n) per ray, n is number of triangles</td>
<td>O(log n) per ray, n is number of triangles</td>
</tr>
</tbody>
</table>

---

**Hierarchy Levels**

**Triangulated Terrain**
Ray Tracing

<table>
<thead>
<tr>
<th>Un-Bundled Rays</th>
<th>Bundled Rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 x 256 array of rays</td>
<td>1 bounding pyramid around the rays</td>
</tr>
<tr>
<td>Each ray is tested for intersection</td>
<td>Pyramid is recursively tested against each partition</td>
</tr>
<tr>
<td></td>
<td>At the leaf partitions, intersect the 256 x 256 rays against the triangles</td>
</tr>
<tr>
<td>O(n), n is number of rays</td>
<td>O(1) partitions, O(n) leaf triangles</td>
</tr>
</tbody>
</table>

Bounding Pyramid

Rays
## Parallel Processing

### Autonomous Landing and Hazard Avoidance (ALHAT)

<table>
<thead>
<tr>
<th>Single Bundle</th>
<th>Parallel Bundles</th>
</tr>
</thead>
<tbody>
<tr>
<td>All the rays in a single bundle</td>
<td>Divide the bundle into sub-bundles, one for each CPU core</td>
</tr>
<tr>
<td>Not easy to parallelize</td>
<td>Independent tasks for 100% parallelization</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Bounding Pyramids**
- **Rays**
Vector CPU Processing

- Modern CPUs are scalar processors
  - Each instruction operates on one data item at a time
- Streaming SIMD Extensions
  - Intel extend the x86 instruction set (SSE)
  - One instruction can operate on
    - 4 32-bit integers
    - 4 32-bit floats
    - 2 64-bit floats
    - 2 64-bit integers
    - 8 16-bit integers
    - 16 8-bit characters
  - Ideal for Vector/Matrix math
  - Additional instructions that must be explicitly used
Emulator Design

Autonomous Landing and Hazard Avoidance (ALHAT)

Simulation Data

Ray Tracer

8-Way Parallel Ray Frustums

Ethernet

Rays

Quad-Tree Terrain

DemMaker
Sensor Model

- Add Gaussian noise to each pixel in the image
  - Signal/Noise Ratio
  - Based on POST2 sensor model or actual hardware characteristics
- Pre-calculate random dead pixels
- Use intensity value for pixel cut-out
- Convolve with Gaussian filter for crosstalk or bleeding between pixels
General-Purpose computation on Graphics Processing Units

- A modern GPU is bigger and has more computational power than the CPU
- Massively parallel, multi-core processor
  - Hundreds of cores per processor
  - Each core is a vector processor
- Ideal for image processing
  - Each pixel will execute the exact same program, in parallel
- Implemented
  - Additive Gaussian Noise
  - Gaussian Convolution
  - Pixel Cut-Out
  - Image Formatting
Emulator Design

Autonomous Landing and Hazard Avoidance (ALHAT)

Simulation Data

Ethernet

8-Way Parallel Ray Frustums

Ray Tracer

Noise Convolution

Signal Processing

Rays

Quad-Tree Terrain

Range/Intensity Image

DemMaker

Camera Link

Command/Control
Application

• Original Problem:
  – How do we develop, test, and evaluate the ALHAT system in a lab environment when we can’t use LIDAR in the lab?

• Field Test
  – All three LIDAR sensors were flown on a helicopter from NASA Dryden
  – The Avionics Processing Box was also flown, collecting data for the GNC and TSAR components
  – The Flash LIDAR was connected to the APB

  – The first time the Flash LIDAR was connected to the Avionics Processing Box
  – The first field test for the LIDAR to integrate image processing and active, intelligent camera control
Application

Autonomous Landing and Hazard Avoidance (ALHAT)

Problem: Although an Interface Documents (ICD) exists, the Flash LIDAR interface has never been implemented. How can we develop and test the interface for a camera when the camera hasn’t been built yet?

Solution: Use the emulator as the testbed to develop the interface
- The ICD and interface needed to be modified for FT4
  - Image header
  - Image resolution
  - System timing
  - Command/control
- The interface was first implemented in the emulator
- The APB was designed, developed, and tested using the emulator interface
- The Flash LIDAR system used the same interface code as the emulator
- The emulator was the first to implement the interface, and all other implementations were based on it, so the emulator became the de facto interface standard
Application

Problem: JSC needs a Flash LIDAR to develop their avionics software. Sending a Flash LIDAR (and person to operate it) to JSC would cost a great deal of time, money, and inconvenience.

Solution: Send an emulator to JSC for their use in software development

- Since JSC didn’t have to wait for the LIDAR to be finished and delivered, The APB and the LIDAR could be developed in parallel.
- The emulator does not require an operator to be with it, so no personnel were required to go to JSC.
- The emulator can be quickly modified for future field tests with very little cost or effort.
Application

**Problem:** It is difficult to develop, test, and debug the image processing and active camera control software using the LIDAR camera.

**Solution:** Use the emulator as the data source for the system software

- The emulator provides a controlled input with known, well-defined values
- The software and LIDAR camera could be developed in parallel
- The emulator can produce data files that can be used to help model and simulate the FPGA code for image processing
- Based on the emulator using the proprietary Ethernet interface, not the ALHAT ICD.
Results
Results

• An emulator was delivered to JSC in August 2009
• JSC used the emulator to develop their APB interface software in preparation for the flight test
• There were no significant interface issues in the flight test, despite the APB and the LIDAR never being physically connected until the field test
• An emulator was used extensively in the development process at Langley. All initial FPGA code was developed and tested using the emulator first
• An emulator was sent to the test site and used for debugging and integration leading up to the test
Future Work

• Integrate the image processing algorithms into the emulator
• Develop an emulator for the velocimeter and altimeter LIDAR systems
• Revise for use in future field tests