2014 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING & SIMULATION, TESTING AND VALIDATION (MSTV) TECHNICAL SESSION
AUGUST 12-14, 2014–NOVI, MICHIGAN

DIGITAL PROXY SIMULATION FOR ROBOTIC HARDWARE

Xi Chen, Ph.D.
Douglas E. Barker, Ph.D.
James A. Bacon
James D. English, Ph.D.
Energid Technologies Corporation
Cambridge, MA

ABSTRACT

Robotics makers and application engineers stand to benefit from replacing physical simulation with a digital simulation that can easily represent any number of robots on a terrain and provide ground truth data for comparison with sensor data during analysis. In this research, a digital proxy simulation (DPS) was developed to dynamically simulate any number of articulated robots in real-time using sophisticated robot-environment interaction models. 3D models of the robot and environment objects can be imported or placed conveniently. Parameters of the models can be fine-tuned to mimic the environment with high fidelity. Sensor simulation and control capabilities of the DPS are also highlighted. Common sensors can be simulated including lidar, image sensors, and stereo cameras. Control plugins can be added easily to accomplish complex tasks.

1 Introduction

It is a common practice to perform field tests in preparation for robotic missions. NASA, for example, tests in remote terrestrial locations before lunar and planetary missions. A robot field test typically involves three groups of people. The first group plans the mission – a process that includes debate among scientists, discussion of the state of the robot, analysis of sensor data, including lidar and visual sensor data (stitched into mosaics) on personal and group-level screens, and prediction of robot performance. The second group controls the operation of the robot. This group transitions between autonomous operation and direct control as needed, and continually analyzes diagnostic data, and monitors communications. The third group travels to a remote location with terrain that matches the needs of the test, such as the desert southwest, or even Antarctica, with robot hardware. This third group monitors the robot, maintains and repairs the robot, and moves the robot to locations that support the needs of scientists. There are typically many obstacles to overcome in modeling mission environments in these tests. Maintaining a group of people that perform a physical test in remote locations may be taxing on the individuals and the supporting institution, and the use of alternate locations may impose some undesirable constraints on field tests.

In these tests, robotics makers and application engineers stand to benefit from replacing physical simulation with a digital simulation that can easily represent any number of robots on a terrain and provide ground truth data for comparison with sensor data during analysis. A digital simulation must provide the following capabilities:

- Easily Configurable Environments: Robots and environmental objects must be easily configurable, as well as sensors mounted on robots or placed in the environments. The digital simulation should be able to import 3D robotic models and provide tools to produce and place environmental objects such as terrains and obstacles.
- Dynamic Simulation: The simulation must include fast physics-based simulations of any number of robots for articulated dynamics, impact dynamics, and wheel-soil interaction that are accurate enough to be either indistinguishable from the real robot or inconsequential.
- Communication Architecture: The digital simulation emulates the robot and environment and must support the same communication protocol of the hardware. For example, NASA maintains software for top-down visualization, sensor-results rendering, mosaic stitching, 3D visualization, and robot tasking. Data Distribution Service (DDS) plays a prominent role in this, and must be supported.
- Sensor Simulations: Robotic vehicles use a variety of sensors to record data, convey information to drivers, and provide diagnostics. These sensors will need to be simulated in software. A special focus will be on lidar sensors, im-

age sensors, and stereo cameras. Modern GPU technologies should be utilized to improve simulation fidelity.

- Control Architecture: The digital simulation should enable remote tasking and local planning for robots.

Several general and specialized simulation tools are available for dynamic simulation, sensor simulation, and control [1, 2]. However, currently there are no simulators that can provide all the above capabilities in one toolkit.

A real-time digital proxy simulation (DPS) was developed based on Energid's Actin product, a general purpose software toolkit for real-time simulating and controlling mechanisms. The robotic hardware and the DPS have the same network interface, such as Data Distribution Service (DDS) for real-time systems. The new proxy simulation seamlessly serves as a replacement for robot communications, actuation, control systems, sensors, environmental interactions, and behavior (Fig. 1). The following sections describes the above capabilities in detail.

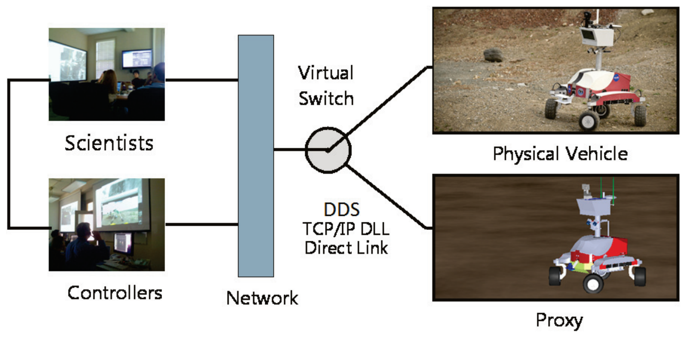


Figure 1: A digital proxy simulation can seamlessly replace robotic hardware for testing, validation, and training.

2 Environment Configuration

The following tools were developed to add objects (robots, rocks, terrains, etc.) to the simulation.

2.1 SolidWorks Converter

The Actin SolidWorks Converter was developed for converting SolidWorks models to the Actin XML format, as illustrated in Fig. 2. The converter creates an XML file with information about the manipulator such as the physical extents (shape and bounding volume) and mass properties (center of mass and moment of inertia) of each link and the joint that connect the links (D-H parameters).

2.2 GeoTIFF Loader

The GeoTIFF loader was developed to load the DEM (digital elevation model [3]) data of a terrain in GeoTIFF formats [4]. The loader also supports loading orthophoto texture image (Fig. 6).

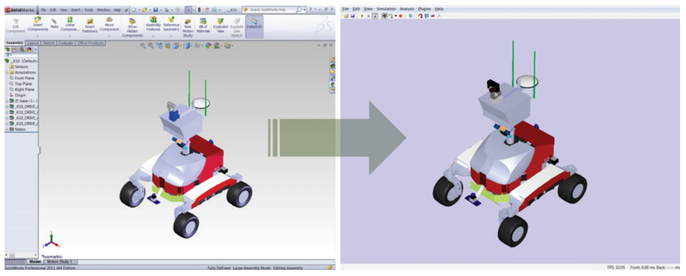


Figure 2: The left picture shows the 3D robot model within SolidWorks and the right picture shows the converted model within the Actin viewer.

2.3 Large Terrain Loading

An important component of the proxy simulation is the ability to load and manage (often large) data sets representing terrain. For this, supports were added to import terrain data from a large number of GeoTIFF files which have been processed into a database using the Virtual Planet Builder (VPB) tool from OpenSceneGraph (OSG) [5]. VPB uses geospatial data abstraction libraries (GDAL) [6] to convert GeoTIFF digital elevation maps and orthoimagery into a database of terrain tiles that can be paged in by OSG at runtime. This makes use the osg::PagedLOD nodes organized into a quad-tree which works in cooperation with the osg::DatabasePager to allow real time rendering of very large terrain datasets (Fig. 3).

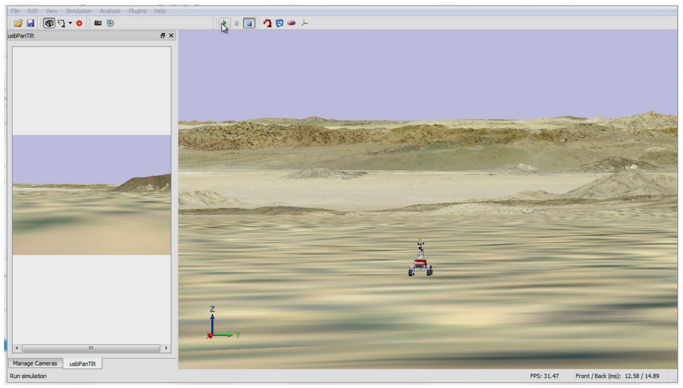


Figure 3: A robot traversing a paged terrain database generated for a subset of the Mojave Desert ($\sim$ 20km range).

For dynamic simulation of the robot on the terrain, a traversable zone is established from which height field data will be loaded to support the robot. In this way the traversable terrain can be just a subset of the visible terrain in order to reduce computational cost and memory footprint. The terrain tiles within the defined traversable zone are managed by the simulation as a union of terrain shapes, where each shape is able to delay loading the height field data until it is required. These shapes contain the physical extents of the tile and path to the height field data.

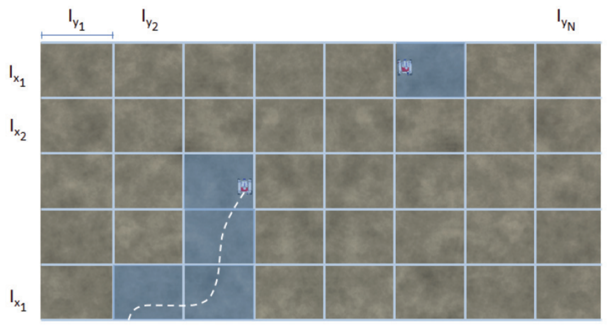


Figure 4: Paging is used to only load in terrain height fields needed for dynamic simulation, while the rendering of a much larger area will be managed by the data base pager and paged level of detail nodes in OpenSceneGraph.

The height field data is loaded as needed based on the location of each robot or dynamically simulated object at runtime. Fig. 4 illustrates a terrain database discretized into loadable terrain tiles.

To quickly determine which terrain tile's height field to load, we employ an algorithm based on intersecting 2D intervals. To do this, the axis aligned bounding box (AABB) around each of the robots and those of each terrain tile are tested for intersection at each time step. The bounding box defines the start and end points for two intervals within the x-y (x = East-West, y = North-South) plane for Universal Transverse Mercator (UTM) projected coordinate systems (Fig. 5). If the robot has come into contact with an terrain tile shape, the height field data will be loaded on demand.

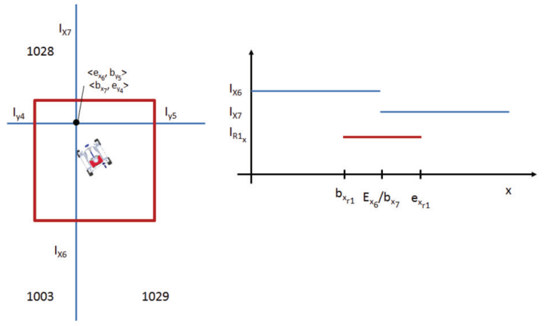
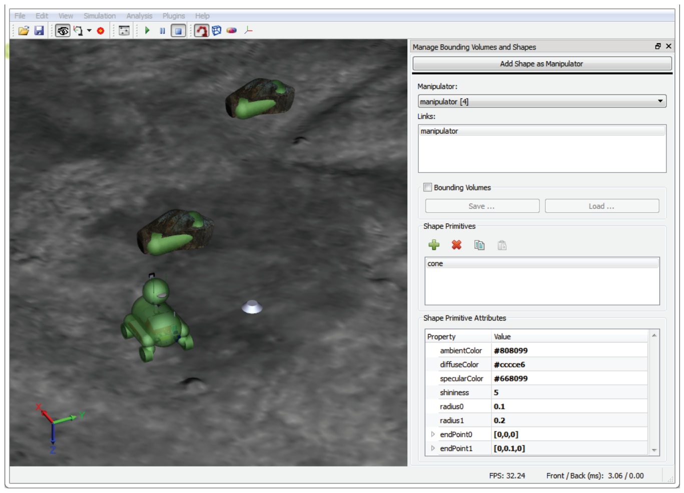


Figure 5: Overlapping intervals in one dimension.

2.4 Shape Primitive Plugin

The shape primitive plugin was developed to change the bounding volumes of each robot or environmental objects or add simple geometric objects to the environment (Fig. 6). The shape of the bounding volume or the object can be an oriented bounding box, tetrahedron, cylinder, cone, sphere, sphere-swept volumes such as capsule and lozenge, or a combination of them.



Manipulator: manipulator [4]

Links: manipulator

Bounding Volumes

Shape Primitives: cone

Shape Primitive Attributes

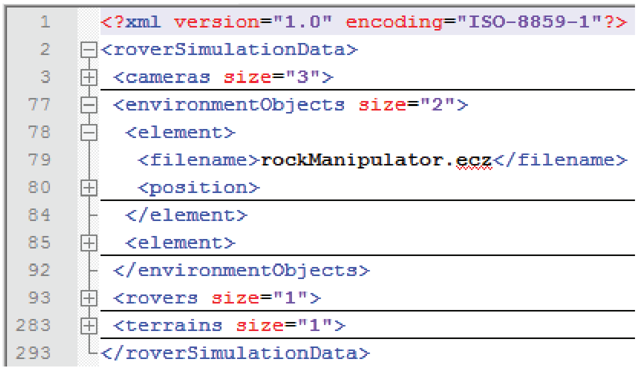
Property	Value
ambientColor	#808099
diffuseColor	#cccce6
specularColor	#668099
shininess	5
radius0	0.1
radius1	0.2
endPoint0	[0,0,0]
endPoint1	[0,0.1,0]

FPS: 32.24 Front / Back (ms): 3.06 / 0.00

Figure 6: A robot on the terrain with bounding volumes shown. A cone shaped object is added using the shape primitive plugin.

2.5 Scene Configuration XML

A scene configuration XML file is used to assemble objects together in a simulation. Fig. 7 shows a sample configuration and Fig. 6 shows the rendered simulation.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<roverSimulationData>
 <cameras size="3">
 <environmentObjects size="2">
  <element>
   <filename>rockManipulator.ecz</filename>
   <position>
  </element>
  <element>
 </environmentObjects>
 <rovers size="1">
 <terrains size="1">
</roverSimulationData>
```

Figure 7: A scene configuration XML file containing a robot, terrain, rocks, and cameras.

3 Terrain-Wheel Interaction Modeling

A terrain object created from a DEM is a 2D height data array mapped to a uniform grid (Fig. 8). The origin, spacing, and number of grid point are defined in the GeoTiff conversion process. For collision detection, the surface between four adjacent grid points is formed by two triangles. Each of the robot's wheels are bounded by a cylinder. The DPS supports penetration depth calculation between any of the shape primitives, and in this case, the penetration depth between a wheel and a terrain is calculated by the penetration depth between a cylinder and a triangle. Then the forces between the terrain and wheels are calculated using the terrain model described below based on the penetration depth.

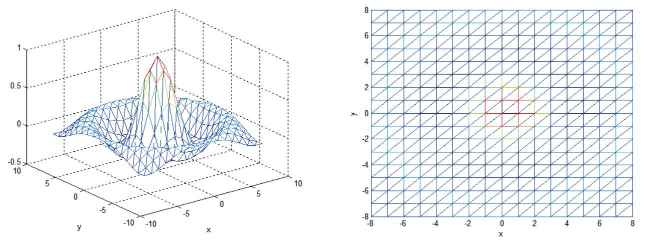


Figure 8: Terrain with the height of a sinc function and triangulation.

3.1 Impact Force

Fig. 9 shows the impact model. A linear spring model used previously was extended to a nonlinear spring, as suggested in [7, 8]. In this model the viscous, or damping, force is proportional to the penetration depth and the total impact force is:

$$\mathbf{f}_{B\to A} = (k_p - k_d v)\,\mathbf{n}_{B\to A} \tag{1}$$

where $\mathbf{n}_{B\to A} = \mathbf{p}_{sA} - \mathbf{p}_A$ is the normal vector, k_p and k_d are surface property constants, and v is the normal speed and is calculated from rigid body kinematics.

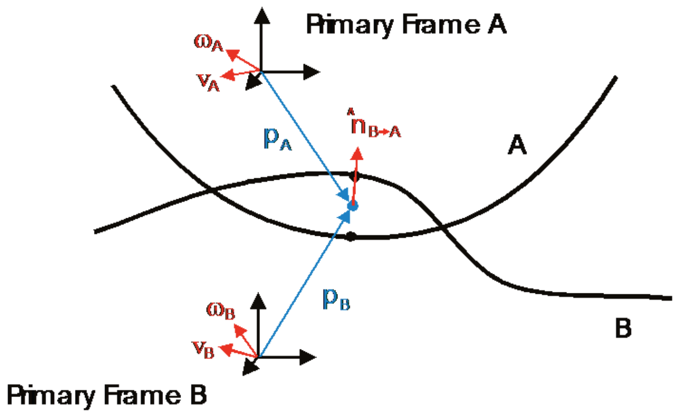


Figure 9: For two intersecting 3D objects, such as a wheel and a rock, the deepest intersection points (support points) are represented in each primary frame as $\mathbf{p}_{sA}$ and $\mathbf{p}_{sB}$; the midpoint between the deepest intersection points is represented in both primary frames as $\mathbf{p}_A$ and $\mathbf{p}_B$.

3.2 Basic Friction

Methods tailored to wheeled motion were added, with a focus on dry friction, which is the most difficult to model [9], but the most useful for terrain interaction. These are based on a breaking spring model of interaction and recent developments in modeling friction through time-stepped simulation. In this, simulation capabilities available in Actin were leveraged. Assuming two objects A and B interact, the Coulomb friction model gives the following constraints:

1. If A rolls and pivots without slipping on B about the contact, i.e., the relative velocity of the contact point $\mathbf{v}_{\mathbf{p}_B/A} = 0$, then for the static friction,

$$\left|\mathbf{f}_{sf}\right| \le \mu_s \left|\mathbf{f}_n\right| \tag{2}$$

where μ_s is the coefficient of static friction, $\mathbf{f}_n$ is the normal force.

2. If A slips on B, i.e., $\mathbf{v}_{\mathbf{p}_B/A} \neq 0$, then the kinetic friction exerted on A is,

$$\mathbf{f}_{kf} = -\mu_k \left|\mathbf{f}_n\right| \frac{\mathbf{v}_{\mathbf{p}_B/A}}{\left|\mathbf{v}_{\mathbf{p}_B/A}\right|} \tag{3}$$

where μ_k is the coefficient of kinetic friction.

For calculating the friction between the two objects, the locations of the contact point in the two reference frames are saved as ${}_0\mathbf{p}_A$ and ${}_0\mathbf{p}_B$ for the first time step that is part of a static-friction mode. This is illustrated in Fig. 10. As the two objects move, this point moves distinctly for the two objects.

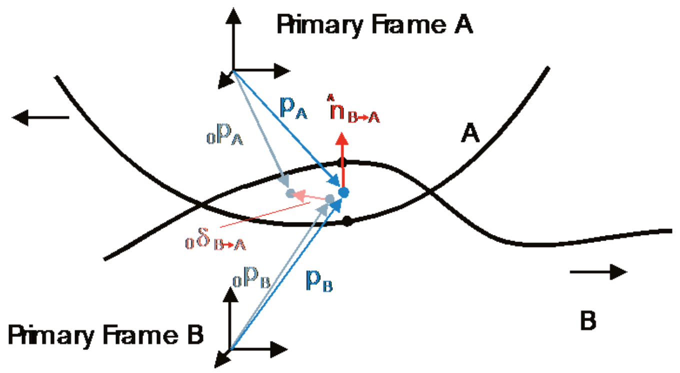


Figure 10: The contact point moves for the two objects. For object A, the contact point moves from ${}_0\mathbf{p}_A$ to $\mathbf{p}_A$ in its primary frame. For object B, the contact point moves from ${}_0\mathbf{p}_B$ to $\mathbf{p}_B$ in its primary frame.

In the following time steps, assuming the static friction condition still applies, the friction force is approximated using a spring-damper response:

$$\mathbf{f}_{sf} = -k_s {}_0\boldsymbol{\delta}_{B\to A} - \lambda_s \mathbf{v}_{\mathbf{p}_B/A} \tag{4}$$

where ${}_0\boldsymbol{\delta}_{B\to A} = \mathbf{p}_A - {}_0\mathbf{p}_A$ is the relative displacement, k_s and λ_s are the spring and damper parameters. The parameter λ_s is calculated as $\lambda_s = 2\hat{m}\sqrt{k_s}$ where $\hat{m}$ is the estimate of m. With this value, the response is critically damped when $m = \hat{m}$.

If $\left|\mathbf{f}_{sf}\right| \le \mu_s \left|\mathbf{f}_n\right|$, the friction is still in static mode; otherwise, the friction is in kinetic mode. In the kinetic mode, the damping part of the static friction is still considered. That is,

$$\mathbf{f}_{sf} = -\lambda_s \mathbf{v}_{\mathbf{p}_B/A} \tag{5}$$

while the kinetic friction is calculated using Eq. (3).

3.3 Advanced Friction

To model interaction between rigid wheel and soil for the robot, we designed an approach using Bekker theory. The inputs to this are the sinkage, the slip ratio, and the slip angle. Given these inputs, this component then computes the drawbar pull ($\mathbf{F}_x$), the side force ($\mathbf{F}_y$), the vertical force ($\mathbf{F}_z$), and the wheel moment ($\mathbf{M}_w$) using a set of integral equations [10, 11]. The force calculations depend on soil characteristics such as cohesion, internal friction angle, cohesive modulus, frictional modulus, shear deformation modulus, sinkage component, and sinkage ratio, and these soil characteristics are added as part of the surface properties. These parameters are then used to calculate forces in the dynamic simulation.

4 Dynamic Simulation

In Actin, all objects (robots, environmental object, etc.) are manipulators. A manipulator is constructed by connecting links in a tree structure, as shown in Fig. 11. Each link stores its mass properties and kinematics such as D-H parameter. This structure supports any number of links and any number of bifurcations. One link serves as the base, with an explicit position and orientation.

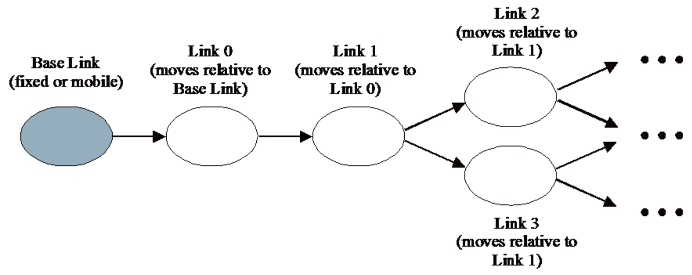

Base Link (fixed or mobile)
Link 0 (moves relative to Base Link)
Link 1 (moves relative to Link 0)
Link 2 (moves relative to Link 1)
Link 3 (moves relative to Link 1)

Figure 11: A manipulator is defined through a link tree. The position and orientation of the base link are represented explicitly. The base link is typically the body of a mobile robot or the base of an arm. For environmental objects (e.g., rocks) the base can be fixed or mobile, as best suited to the scenario.

A dynamic simulation numerically integrates Newton's and Euler's dynamic equations to model moving parts. Articulated dynamics, impact dynamics, and the dynamics of the motor controllers are included in this calculation. For calculating the dynamics of articulated mechanisms, we use the composite rigid body algorithm [12, 13] for small manipulators (less than 12 degrees of freedom). This algorithm runs in $O\left(N^3\right)$ time, for N links, based on the following equation:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{D}\mathbf{A}_b + \mathbf{B} \tag{6}$$

where $\boldsymbol{\tau}$ is the column vector of joint torques/forces, $\mathbf{M}(\mathbf{q})$ is the manipulator inertia matrix, $\mathbf{q}$ is the vector of joint position, $\mathbf{C}(\mathbf{q})$ represents the Coriolis forces, $\mathbf{G}(\mathbf{q})$ represents gravitational forces, and $\mathbf{B}$ represents the effect of external forces applied to the arms links. $\mathbf{A}_b$ is the acceleration of the base link. $\mathbf{D}$ is a special dynamic matrix that is a function of configuration only. Combining this equation with direct application of Newtons and Eulers laws to the base link allows solution of $\ddot{\mathbf{q}}$ and $\mathbf{A}_b$ for mobile systems through the solution of an $(N+6)$ degrees-of-freedom (DOF) system of linear equations.

The articulated-body algorithm [14, 15] is used for more complex manipulators, running in $O(N)$ time. Actin is capable of running faster than real-time, though in the proxy simulation, it runs in real-time with frame rate no less than 30.

5 Environment Rendering and Sensor Simulation

5.1 Lidar Simulation

Assuming that a 3D model of a scene exists, lidar scans can be simulated by casting rays into the scene and measuring the distances of the intersecting points; however, this is time consuming, and in this work we utilized an efficient OpenGL based approach.

5.1.1 OpenGL Approach

Instead of casting one ray at a time, an OpenGL approach was used to gain dramatically in speed (by more than two orders of magnitude) by performing computations on the computers graphics card. A concern was that OpenGL would be challenged in modeling accurate distances over large ranges. However, most modern (low- and mid-end) GPUs can support 24-bit depth buffer. Some high-end GPUs can even support a 32-bit depth buffer. The analysis given below shows that OpenGL with 24-bit depth buffer is adequate to model lidar data.

The depth buffer (also known as z-buffer) is used in OpenGL to resolve the distance between two nearby objects to determine the proper rendering order for overlapping objects. Because the z-buffer values can be easily mapped to physical distances, it is almost directly applicable for simulating lidar scanners.

The near and far clipping planes are the planes defining what objects will be rendered in the scene. Only the objects located between the two planes will be rendered. *zNear* and *zFar* are the distances from the eye to the near and far clipping planes, respectively. For synthetic lidar scanners, *zNear* and *zFar* can be thought of as the minimum and maximum range values of the scanners.

The z-buffer is nonlinear. The actual number stored in the z-buffer memory can be expressed in terms of the distance to the object as

$$z = (2^N - 1) \cdot \left(a + \frac{b}{d}\right) \tag{7}$$

where N is the number of bits of Z precision, d is the distance from the eye (sensor) to the object, and

$$\begin{aligned} a &= \frac{zFar}{(zFar - zNear)} \\ b &= \frac{zFar \cdot zNear}{(zNear - zFar)} \end{aligned} \tag{8}$$

From Eq. (7), one can observe that z trends inversely proportional to d, and hence the precision is better for objects closer to the eye point than those farther away. To determine the resolution, we denote two successive z values with z_1 and z_2 ($z_1 - z_2 = 1$) and the distances at those z values be d_1 and d_2. It follows from Eq. (7) that the smallest discernible distance δ is given by

$$\delta = -\frac{d_1 d_2}{(2^N - 1)b} \approx -\frac{d^2}{(2^N - 1)b} \tag{9}$$

At $d = 350$m, $zNear = 3$m, and $zFar = 350$m, using the above equation, the resolution for 24-bit z-buffer is 0.002413 (or 2.4mm). This compares favorably with the raw range accuracy of 7mm at 100m in the lidar specifications and the fact that the range data is 16-bit integers (which translates to 5.34mm discretization over 350m range if linear). Based on this analysis, the OpenGL depth buffer approach is adequate for most anticipated purposes even when using only 24 bits.

There was an additional consideration for OpenGL-based lidar modeling. A 3D perspective view of a scene in created through a view frustum (Fig. 12). Anything inside the frustum is rendered. A distance value is calculated as

$$d = \frac{b\,(2^N - 1)}{z - a\,(2^N - 1)} \tag{10}$$

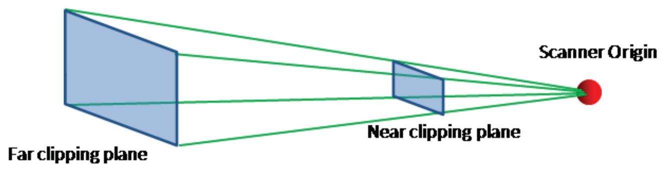


Figure 12: The view frustum (volume between the near and far clipping planes) and depth buffer were used for high-speed simulation of lidar scanners.

The distance obtained from Eq. (10) is measured perpendicular to the near clipping plane. If used directly as the scanner distance, it will cause hemispherical distortion; i.e., it will cause the points on a plane to look as if they were on a hemispherical surface. Instead the ratio between the distance at each scan point on the near clipping plane and the normal distance must be computed. This ratio is called the stretch factor and is given by

$$s_i = \frac{dNear_i}{zNear} = \sqrt{\tan^2(\theta_i) + \tan^2(\phi_i) + 1} \tag{11}$$

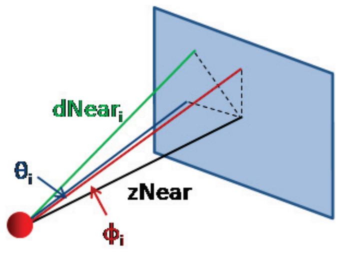


Figure 13: Quantities involved in computing the stretch factor.

where θ_i and ϕ_i are the horizontal and vertical angles of ray i, respectively (Fig. 13). The stretch factor is multiplied with the initial distance to yield the distortion-corrected distance:

$$d_i = \frac{b\,(2^N - 1)\,s_i}{z - a\,(2^N - 1)} \tag{12}$$

5.1.2 Noise

Two types of lidar noise are modeled, longitudinal and orthogonal. Longitudinal noise represents the uncertainties in the distances along the rays and affects range accuracy. Orthogonal noise corresponds to the uncertainties in the directions of the rays and affects position accuracy. In this work, both were generated using zero-mean Gaussian distributions. Each has its own variance (standard deviation). Longitudinal noise was included by adding a random value to the distance of each ray, and orthogonal noise was included by adding a random orthogonal vector. The result of lidar simulation is shown in Fig. 14.

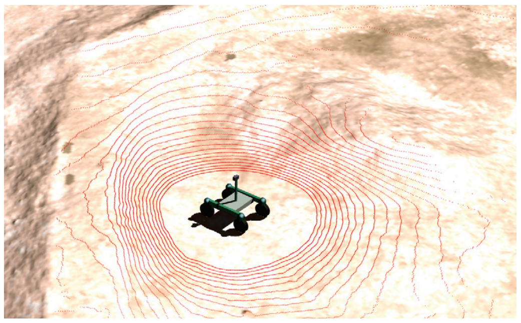


Figure 14: Lidar simulation.

5.2 Lighting and Shadows

Accurate sensor models that include realistic lighting, shadows, and sensor noise are vital to the framework we propose. In this work, lights can be added, removed, edited, and copied with a simple light source manager widget. Lights can be placed with respect to the global coordinate system or attached to any mechanism or link in the simulation. Directional and positional lights can be easily configured in the simulation using the dragger tool in the main viewer window. This allows users to quickly reposition a light and see the scene update in real-time.

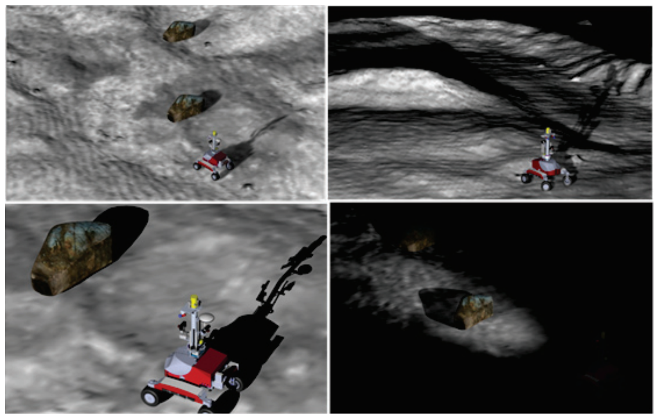


Figure 15: Real-time shadows rendering in Actin: a) with directional lighting with soft shadow mapping, b) positional lighting with soft shadow mapping, c) light space perspective shadow mapping with directional lighting, d) spot light attached to the robot used for shadowing.

Shadow rendering was incorporated into Actin's rasterization pipeline. The images in Fig. 15 are examples of the new shadow rendering capability. The two variants of the Shadow Map technique implemented, soft shadow mapping and light space perspective shadow mapping, are shown in images a) and c) respectively. The light space technique produces very crisp and dark shadows in the view port of the camera, and is best used with direct lighting, while the soft shadow map produces faint diffuse shadows. Shadows can be cast by all objects and received by all objects. Fig. 15 (d) shows a positional spot light attached to the robot to simulate a headlight, and used to cast shadows as seen by the rock on the terrain. The primary purpose of shadow rendering is to support the simulated cameras.

5.3 Camera Simulation

To create cameras, Energid leveraged capability in Actin. Any number of cameras can be rigidly attached with any position and any orientation to any link on the robot. Each camera can be assigned any field of view and any size in pixels, and noise can be added to simulate realism. A simulation scenario of shadows rendered with directional light using soft shadow mapping with three on-board cameras being illustrated through pop-up windows is shown in Fig. 16.

5.4 Stereo Cameras

To simulate stereo cameras, two or more standard camera images are rendered synthetically and stereo algorithms from OpenCV are applied to the result. High-fidelity rendering with texture and lighted features enables this to give excellent results, as shown in Fig. 17. Though this approach is potentially slower than using depth information directly to simulate results, it is more accurate, as it captures the loss of information and errors that arise from loss of contrast and apparent texture in the images.

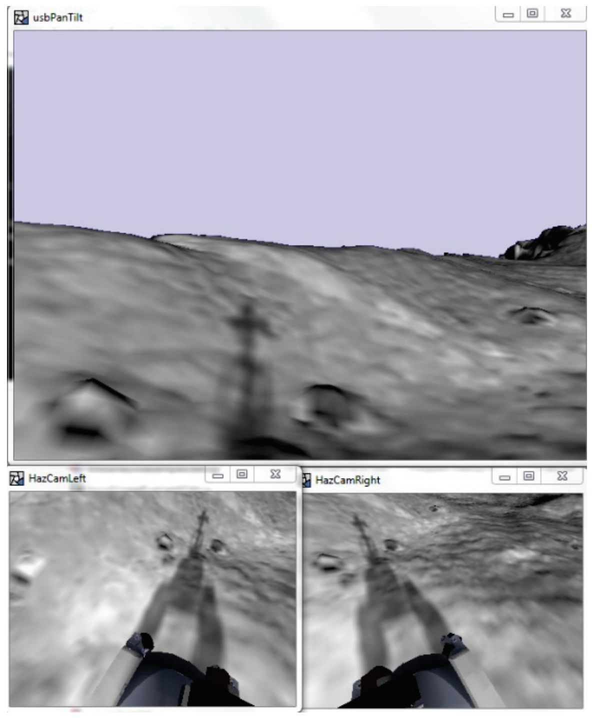


Figure 16: Camera views onboard the robot using a soft shadow map and directional lighting.

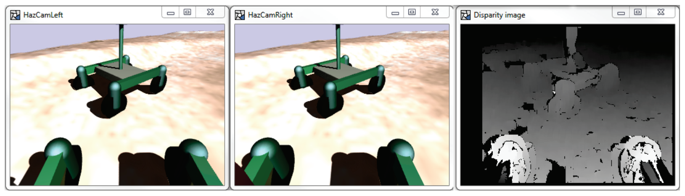


Figure 17: Left, right, and disparity images of a stereo camera.

5.5 Ray Tracing

Ray tracing is a method of creating physics-based photo-realistic images of a virtual scene by tracing each ray of light from sources through objects and to the eye of the viewer. Rays hitting objects can be scattered, reflected or refracted. Due to its computational expense, traditionally this has been an offline process completed by the CPU over several minutes. But now graphics cards and parallel programing paradigms have advanced to allow completion of this task in milliseconds. We created an NVIDIA Optix camera plugin for the Actin simulation that reads in the scene graph and ray-traces the scene from the perspective of the Actins main viewer window. The objective was to establish the core infrastructure to convert the Actin scene graph into a form that works with Optix, and show that real-time ray tracing is possible for the proxy simulation. The mapping between Actins scene graph and Optix was defined in this work and the converted result is shown in Fig. 18.

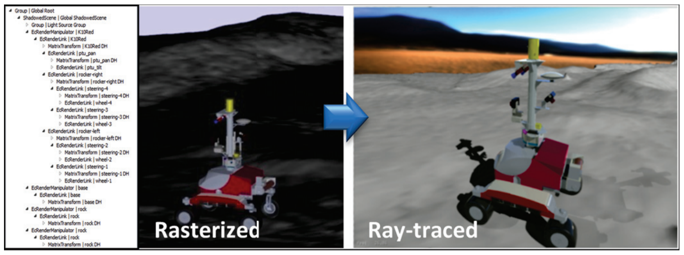


Figure 18: The basic scene graph conversion from the earlier version of Actin to Optix was defined in this work. Note the multiple shadows coming naturally from two light sources in the ray-traced image.

Figure 19: Real-time rendering of reflective terrain surface properties to demonstrate reflections from the environmental map onto surfaces of the robot.

True reflections are faithfully reproduced using ray tracing. Fig. 19 shows an example of changing the material attributes for the terrain from non-reflective to reflective. Note that the terrain surface appearance is now determined by the light reflected off of the environmental map and the robot. The time performance of the ray-tracing developed for the proxy simulation is provided in Fig. 20. NVIDIAs GeForce 640M notebook graphics card and the professional level Quadro K5000 were tested. With over 2M rays cast, the K5000 was able to easily process the scene in real-time at 20 frames per second. Even with the less capable 640M, a respectable 5 frames per second was achieved.

6 Control

It is very convenient to write control algorithms for robots in Actin. In this work, a path planning and control algorithm was implemented to support application to lunar polar missions, where terrain shadows can significantly reduce power availability and paths should be as illuminated as possible.

Illumination must be combined with other factors, and we implemented a path planning module that builds an occupancy grid map to store various cost due to differences in height, obstacles, and shadows. It uses A* search to plan a path with minimum cost. A controller was designed specifically for the robot to drive it to follow this path.

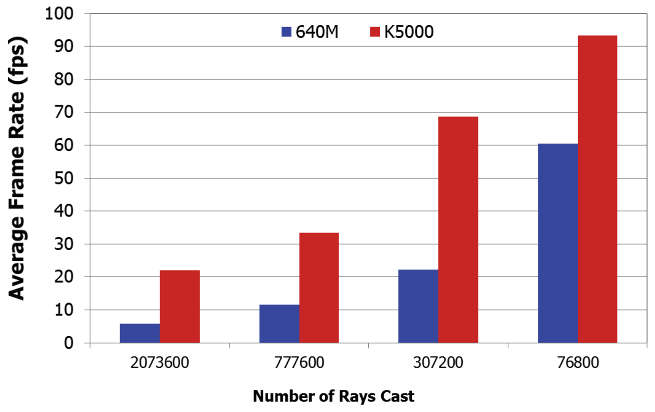


Figure 20: Ray-tracing performance of two NVIDIA graphics cards on the proxy simulation scene ($\sim$ 1.5M polygons), showing the potential for using real-time ray tracing for sensor rendering in the randomizing-optimizing simulation.

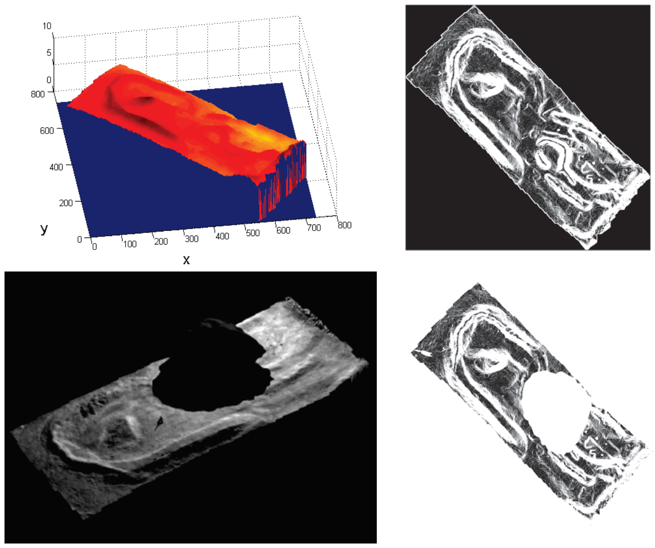
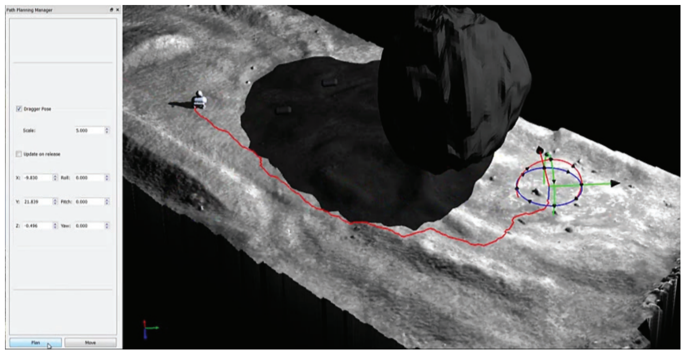


Figure 21: (a) height field of a terrain. (b) calculated cost due the terrain gradients. (c) a representative boulder on the terrain with shadows. (d) final cost map incorporating slope and shadows.

The cost map for a terrain is a two-dimensional matrix of costs. To build the grid map, a terrain is loaded first which defines heights at each grid nodes as shown in Fig. 21 (a). Fig. 21 (b) shows the height gradient image **G**, which measures how steep the terrain is, calculated from Fig. 21 (a). The whiter the pixel at each grid point is, the higher the cost it represents (steeper). Once the shadow is rendered, as in Fig. 21 (c), we can get an image of the scene and calculate the cost map **S** due to the shadows by analyzing the pixel intensities of the rendered im-



☑ Dragger Pose
Scale: 5.000
☐ Update on release
X: -9.830 Roll: 0.000
Y: 21.839 Pitch: 0.000
Z: -0.496 Yaw: 0.000

Figure 22: An illuminated path to the goal.

age. The final cost map is a weighted sum of the height gradient **G** and shadow **S**. Fig. 21 shows the final cost map.

In this work, the optimal path was found by A* search and was optionally rendered in Actin so that one can qualitatively assess it. Fig. 22 shows an illuminated path to a user specified goal using the cost map of Fig. 21 (d).

A gradient based path following algorithm developed by the author before [16] was used to follow the planned path and a PID controller was designed specifically for the robot to send torque commands to wheel actuators.

7 Conclusion

The DPS was successfully applied to dynamically simulate NASA rovers, military vehicles, research vehicles, and commercial exploration robots in real-time. Objects in the environment can be easily configured. Interactions between objects and sensors are simulated with high fidelity. Control algorithm can be added conveniently. The DPS fills the need to seamlessly serve as a replacement for real hardware and can be applied to simulate any articulated robotic vehicles with complex sensors and control systems in complex environments.

References

[1] Robot Operating System, http://wiki.ros.org/gazebo

[2] J. Cameron, S. Myint, C. Kuo, A. Jain, H. Grip, P. Jayakumar, J. Overholt, “Real-time and High-fidelity Simulation Environment for Autonomous Ground Vehicle Dynamics,” Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), Troy, Michigan, August 20-24, 2013.

[3] Z. Li, Q. Zhu, C. Gold, “Digital terrain modeling: principles and methodology,” CRC Press, Boca Raton, 2005.

[4] Sk. Sazid Mahammad, R. Ramakrishnan, “GeoTIFF-A standard image file format for GIS applications,” http://www.gisdevelopment.net/technology/ip/mi03117pf.htm

[5] http://www.openscenegraph.org/

[6] http://www.gdal.org/

[7] D.W. Marhefka, D.E. Orin, “A Compliant Contact Model with Nonlinear Damping for Simulation of Robotic Systems,” IEEE Trans. on Sys., Man, and CyberneticsPart A: Systems and Humans, vol. 29, no. 6, pp. 566-572, 1999.

[8] K. Yamane, Y. Nakamura, “Stable Penalty-Based Model of Frictional Contacts,” Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida, May 2006.

[9] D. Baraff, “Coping with Friction for Non-Penetrating Rigid Body Simulation,” Siggraph ’91, Las Vegas, vol. 25, no. 4, pp. 31-40, 1991.

[10] J.Y. Wong, “Theory of Ground Vehicles,” John Wiley & Sons, 2008.

[11] K. Yoshida, G. Ishigami, “Steering Characteristics of a Rigid Wheel for Exploration on Loose Soil,” Proc. IEEE Int. Conf. on Intelligent Robots and Systems, Sendai, Japan, 2004.

[12] M.W. Walker and D.E. Orin, “Efficient Dynamic Computer Simulation of Robotic Mechanisms,” Journal of Dynamic Systems, Measurement, and Control, 104, 205-211, 1982.

[13] A. Fijany and A.K. Bejczy, “An Efficient Algorithm for Computation of Manipulator Inertia Matrix,” Journal of Robotic Systems, 7(1), 57-80, 1990.

[14] R. Featherstone, Robot Dynamics Algorithms, Kluwer Academic Publishers, Boston, 1987.

[15] K.W. Lilly, Efficient Dynamic Simulation of Robotic Mechanisms, Kluwer Academic Publishers, Boston, 1993.

[16] X. Chen, C. Ragonesi, J.C. Galloway, S.K. Agrawal, “Training Toddlers Seated on Mobile Robots to Drive Indoors Amidst Obstacles,” IEEE Trans. Neural Systems and Rehabilitation Eng., vol. 19, no. 3, pp. 271-279, June 2011.