Havok Game Dynamics SDK

*A **product overview** of the most comprehensive, optimized, real-time physics solution for game development.*

Contents

1 Product Summary

The Havok Game Dynamics SDK is a comprehensive highly optimized development solution designed to simplify the task to integrating full physical simulation into entertainment applications.

The core technology has support for the following:

- Optimized collision detection & resolution
- Rigid body response
- Deformable dynamics: soft body (mesh and FFD based), cloth and rope
- Vehicle Dynamics
- Fast general constraints
- Specific car and rag doll constraints
- Sensors, events & callbacks
- Rapid prototyping layer
- Debugging and profiling support
- Demos, tutorials and documentation

In addition to the core engine, the Havok Game Dynamics SDK features a comprehensive set of tools for programmers, artists and designers to facilitate parallel development streams early in the project lifecycle and easy tuning and profiling of your game.

- Exporters for 3ds max and Maya, for the easy set-up of physical scenes within a familiar environment
- Car tuning tool, for tweaking over 100 vehicle parameters, while testing vehicle drivability in real-time on your target platform
- Visual Debugger, for analyzing and profiling your game

Supported Platforms

The Havok 1.7 Game Dynamics SDK is available for PlayStation 2, GameCube, Xbox and PC and supports all commonly used compilers (e.g. Codewarrior, ProDG). Havok's core technology is optimized for each supported platform.

Havok also supports all 3D graphics engines: Alchemy from Intrinsic, NetImmerse from NDL, Renderware from Criterion, the Quake and Unreal engines, as well as all in-house rendering engines.

2 Collision Detection

Collision geometry types supported by Havok are:

- Spheres and Planes
- Convex objects
- Concave objects (not necessarily closed)
- Fixed polygon soups (e.g. landscapes): used for geometry, which never moves and has been optimized for this purpose.
- Heightfields (2 flavors: optimized and unoptimized).

	Convex	Concave	Sphere	Plane	PolySoup	Heightfield	Optimized Heightfield
Convex	✓	✓	✓	✓	✓	✓	✓
Concave	✓	✓	✓	✓	✓	×	×
Sphere	✓	✓	✓	✓	✓	×	×
Plane	✓	✓	✓	×	×	×	×
PolySoup	✓	✓	✓	×	×	×	×
Heightfield	✓	×	×	×	×	×	×
Optimized Heightfield	✓	×	×	×	×	×	×

The matrix above details the collision detection implementation in the current release. Each square represents whether collision are calculated for the associated features. In some cases (e.g. planes colliding with planes) the collisions are purposefully not implement. Currently, the height-field support is limited to rigid convex objects but support for all other object classes is in development.

Feature Benefits:

Feature	Description	Benefit
Optimized Collision Detection Engine	The Havok Collision Detection Engine requires significantly less memory than any other implementation available on the market today.	Havok has developed a new mid-phase collision detection system based on Memory Optimized Partial Polytope (MOPP) Technology. The revolutionary MOPP Technology significantly reduces the amount of memory required to store collision detection information, making it possible to create large physical landscapes for the first time on platforms like PS2 and Gamecube. An added bonus is that the new technology also improves game performance in many instances. In particular, vehicle games that use raycasts will benefit the most from these improvements.
Access to collision results	Each collision gives a position, normal, relative velocity and distance from surface information.	Gives you total control of the effects you want to create. For example, you can change the volume levels of explosions depending on the relative velocity of the objects colliding.
User defined collision primitives	The Havok SDK features support for user defined collision primitives. This provides a callback during collision detection that can be used to add your own collision types.	Havok gives you the flexibility to create your own collision primitives. Use of collision primitives optimizes performance in your game.
Query collision system	The collision system may be called independently of the physics, by disabling collision response.	You will often want to detect collisions without calling the physical responses into play. We give you the freedom to turn off the collision response.
Display proxies	Represent any object in the game by an arbitrary collision geometry.	Improve your game engine's performance – by decoupling display and collision shapes, you can display something relatively complex while simulating it as a box or sphere. In this way, you get control of the CPU used by the collision detection in their game.
Phantom Objects	Objects which are included in the collision detection process but do not cause collisions.	These are great for making sensors: for example, a box phantom object might be placed surrounding a room – if an object penetrates the box a collision is registered and information passed to the client, but the object does not bounce off the box – instead it passes straight through.
Collision Filtering	All collision events may be filtered according to game specific needs.	This allows you to optimize the collision detection for your game – you can remove collisions from the system where appropriate.
Collision Layers	Objects in the system may be grouped into layers, and the collisions between layers disabled or enabled.	This gives you even more control over the collision system allowing you to further optimize performance. For example, objects in different rooms might be gathered into different layers, and collisions between these layers disabled.

3 Simulation & Collision Response

The collision response subsystem manages what happens to objects after they collide. This system is responsible for sliding, bouncing, stacking and general movement of all the objects in the physical simulation.

Feature Benefits:

Feature	Description	Benefit
Physical Properties	Physics properties like mass, elasticity, friction and restitution can be set for each rigid body. You can also change the center of mass.	At any time in the game any of the physics properties can be set or changed giving full control over the behavior of the objects. Some physics engines require that you fix the properties of the objects in the scene at scene construction time – this is not the case with Havok.
Fast Subspace	A new collision response algorithm may be selected for cases where you want low CPU hit and are prepared to sacrifice some accuracy.	Fast subspace is great for rockfalls, meteor storms etc, where you don't always see the results of a slightly less accurate physics simulation, but want to simulate lots of objects.
ODE Solvers	A selection of ODE solvers[1] is provided including Euler and RK45.	If you want to have greater control over how the simulation is handled numerically you can specify the integrator to use
Multiple Friction Models	There are a number of friction models provided with the Havok SDK, from simple to complex (handling both static and dynamic friction).	Depending on the accuracy you want in the simulation you can choose which friction model to go for. This gives you greater control over the CPU hit required for the simulation. The complex friction model seamlessly handles static/dynamic friction, which is a requirement for stacking and sliding.
Stable Stacking	When lots of objects are in contact simultaneously, the Havok SDK handles this to provide stable stacks as you would expect (i.e. objects don't continue to slide about).	Without stable stacking it's hard to create structures like walls, or place things on tables etc. Stable stacking is a necessary feature to provide realism in a physical simulation.
Math Library	A fully featured math library with support for vectors, matrices and Quaternions is provided with the Havok SDK.	You can use the math library for your own purposes, building on our stable and optimized routines for your own game. These libraries have been well tested as they form the core of the Havok SDK.

[1] An ODE is an ordinary differential equation (in this case the equation is that which describes the motions of the rigid bodies). An ODE solver is a piece of math code that solves the equations, and in this case works out how the objects move in the simulation.

4 Scene Management

The Havok SDK provides a set of functions to allow you to manage the simulation of your game world in a scalable way, allowing it to handle large geometries and large numbers of dynamic objects.

Feature Benefits:

Feature	Description	Benefit
Dynamic Creation & Deletion	All objects can be created and deleted at runtime, adding and removing them to the physical simulation.	This is great for large worlds where you only want those objects close to the player to be active in the simulation and taking up memory. When they get too far away, just delete them.
Subspaces	Havok groups simulated entities into subspaces. Each subspace can be managed separately, with their own integrators, deactivation parameters etc.	You can break your world up into subspaces and then manage the CPU load across these subspaces in real time. For example, a subspace that is not near the player might be updated less frequently.. In this way you can better manage the CPU load.
Cloning	Physical objects may be cloned, having a single memory resident version of the geometry but as many instances as you want.	If you have many repetitions of an object (rock in a rockfall, leaves in the wind etc.) then you only need to store one geometry and then instance it many times, so you have very efficient memory usage.
Automatic Deactivation	Objects that are not moving are removed automatically from the simulation. They are not deleted, just tagged as "inactive" and do not wake up until collided with or explicitly activated through the SDK. The user has control over how aggressive the deactivator is so that it may be tuned to a specific application.	In most cases only a few objects at any given time are truly active and moving in the scene. The automatic deactivation optimizes CPU usage by removing inactive objects. With the tuning parameters you can deactivate objects aggressively (e.g. in a rockfall where you won't notice) or more subtly.

5 Constraints

Havok has a fully featured constrained dynamics engine tailored for real-time game applications. Our constraints have been optimized for games and are extremely efficient while very stable. All constraint properties and connections are editable in real-time so you can make or break constraints at will at run-time.

Feature Benefits:

Feature	Description	Benefit
Point to Point	Constrain a point on one object to a point on another with full joint limiting and joint friction.	Fast and stable point to point make it easy to connect objects together (great for effects like ropebridges, machinery, hanging lights etc).
Point to Nail	A special case of point to point allows you to connect an object to a fixed point in space, again with full joint limiting and joint friction.	Good for anchoring objects to parts of the landscape.
Joint Limits	You can specify for most constraints the limits of the relation motions and orientations of the constrained bodies.	Allows constraints like hinges to only twist through a limited angle, or for rag doll limbs to ensure that they always maintain realistic poses.
Joint Friction	Joints with friction lose energy as the joint is manipulated, so an arm joint will not result in an arm that spins forever.	Really important to allow rotations around constraints to come to rest eventually
Hinge joint	Allow one object to spin around the axis of another.	Great for doors and levers like seesaws.
Rag Doll constraints	A special constraint designed to match the sort of connections that exist between limbs in humans. There are soft joint limits and friction with independent control over positional and angular constraints.	This makes it very easy to create rag doll effects, like bodies falling having been killed or a motorcyclist reacting to bumps in the road or passengers in a car as it powerslides around corners.
Car Wheel constraints	A special purpose constraint to make it easy to construct vehicles. A wheel may be constrained to spin about an axel while also having limited rotational spin around a steering axis.	Although we have a vehicle SDK that replaces wheels with ray casts, sometimes you want to construct a simple vehicle (e.g. trailer). The Car wheel constraint is great for this. It's also possible to create cars and other vehicles with this constraint (and use springs for suspension), but this is not as efficient as using out car SDK, but may be more general in some cases.
Stiff Springs	A faster alternative to point-to-point constraints. Stiff springs are very efficient to computer but are not as stiff as constrained joints.	Great when you want to connect things (and lots of them) but are not as concerned about a little bit of drift in the connection (i.e. the connected objects may move apart a little under enough force).
Springs	Classic Hookean springs with control over rest length (compression and extension) and damping.	Sometimes a spring is just what you need! Good for joint actuators or suspensions.

Feature	Description	Benefit
Dashpots	Dashpots are similar to springs but act on velocities of objects not forces. They may be used when springs are not stiff enough, and where stiff springs are too expensive to compute. Dashpots come in linear and angular flavours.	Dashpots are great for creating constrained systems very cheaply. It's easy to create a rag doll effect which uses little CPU but which is perhaps not as accurate or controllable as one created with rag doll constraints.

6 Vehicle Dynamics

The Havok vehicle dynamics module is made up of a core vehicle physics system with efficient ray casting for wheels and a tuned car wheel friction model to allow all types of vehicle behaviors to be simulated effectively. On top of this core are a series of game specific modules, which are available with source (and thus are user replaceable). These include modules for:

- Transmission
- Engine
- Suspension
- Gameplay
- Aerodynamics
- Steering

The default system provided (complete with car tuning tool) gives access to over 100 parameters all of which may be tweaked in real-time for outstanding vehicle design control. Some of these parameters are:

- *Wheel parameters*: radius, width, friction, axle
- *Suspension parameters*: length, strength, damping, suspension hardpoint
- *Engine parameters*: rpm, torque, resistance torque
- *Gearing parameters*: gear levels, shift points, clutch delay, wheel torque ratios
- *Aerodynamics parameters*: air density, drag coefficient, lift coefficient
- *Gameplay parameters*: friction equalizer, inertia factors, wheel friction tweakers

The vehicle SDK is based on a 2-axel steerable model with an unlimited number of wheels per axel.

Feature Benefits:

Feature	Description	Benefit
Car Tuning Tool	A real time UI giving direct access to most of the vehicle parameters using sliders and other controls. The UI can communicate directly with the target device (e.g. PS2) to allow you to tweak the vehicle for your game. See the tools section for more details.	Allows exploration of the functionality of the vehicle SDK without having to create your own application. This is great way to get familiar with the vehicle SDK features and also to make a stab at the parameters to extract a particular type of behavior from the vehicle technology. This can be can be useful for game designers to get a feel for the cars they create, and save time when prototyping vehicle dynamics. Also, given that source is provided, you can extend this into a specific tool for your own car game. The on-target nature of the tool means that you can test vehicle performance on consoles like PS2 and XBox directly.
Modular with Source	The vehicle system has been designed as a core physics module with all functional components separated into distinct modules. All of the application specific vehicle modules are provided with full source (e.g. engine, transmission, aerodynamics).	All application specific modules can be altered to directly suit the requirements of your game. If you want to implement your own strange transmission method then you're free to do so. This makes the vehicle SDK a truly open and extensible framework for vehicle dynamics simulation.
Raycasting for Wheels	Although wheel constraints are provided in the core SDK, the vehicle SDK uses fast ray casting for simulating the suspension/wheels of the vehicles. The ray cast takes full account of wheel radius and width.	Raycasting for wheels gives superior performance across all supported platforms with negligible visual loss in quality. You can specify the wheel width, so the ray casting performs accurately even when the wheels come in contact with low edges like kerbs.
100+ parameters	There is a vast array of parameters associated with each of the vehicle modules, with full real-time access always available to change the vehicle performance during game play or game design.	Every aspect of the vehicle system is tweakable to extract just the sort of behavior you are looking for. With full real time access you can test the parameter changes in-situ, as you design the game. And with our modular approach with full source you can create your own parameters or entire vehicle modules to suit a specific requirement.
Skidmarks & Camera Support	Support is provided through an event system for tire tracks and skid marks. We also provide techniques for dealing with cameras following the vehicle.	The visual representation of the vehicle has a huge impact on the perception of vehicle performance and speed. We provide graphical methods to extract best use of the vehicle dynamics, from skid marks to camera techniques that enhance the appearance of speed.
Advanced wheel friction model	An isotropic model of friction for each wheel is provided to allow modeling of many different types of tire and ground surfaces.	A good tire friction model is crucial for achieving accuracte over-steer, under steer, powersliding, handbrake turns etc. Our model captures many of the physical features of modern radial tires as well as being able to model many other less "standard" tire types.

7 Fast Deformable Technology

Fast deformable technology provides fast cloth, soft body and rope dynamics with control over stiffness and damping and extremely efficient collision detection based on deflectors. With this technology, you can easily tweak the parameters to optimize your game's performance.

Feature Benefits:

Feature	Description	Benefit
Fast Integration	A fast integration method is used for the fast deformable technology.	The deformable technology is very efficient taking little CPU time to simulate complex deformations.
Deflectors	A new approach to collision detection has been provided based on deflectors or collision regions that may be setup by the user. Each vertex in the simulation can be instructed to collide with any number of deflectors.	The collision performance of the deformable object may be completely tailored to suit the application. By having vertex level control over collision detection you can achieve very efficient collisions with little loss in visual accuracy.
Attachment Constraints	Every deformable body may be attached to another or to rigid elements in the scene through a series of constraints. There are no limits of these constraints and they may be made and broken at runtime. Constraints may be specified at a vertex level for complex attachments.	The deformable constraint system allows you to construct complex scene elements in a straightforward manner. Cloth may be attached to moving characters, soft portions may be attach to rigid frames, and ropes can be attached to cloth / soft for effects like tassels.
Volume Constraints	Specified vertices in the deformable object may be constrained to lie inside a volume specified by a deflector object.	This gives you really fine grain control over the behavior of the cloth. Sometimes you'll want to ensure that parts of the cloth do not move too far (for example to approximate a loose belt around a piece of clothing.)
Full max exporter support	Creation of deflectors, constraints and deformable bodies is fully supported in the max exporter.	Although a large number of complex parameters are made available through the deformable technology, the max exporter gives an intuitive and artist-friendly interface to the technology making it simple to integrate into an existing game production framework.

Deformable Bodies Parameters:

- Common properties:
  - Mass
  - Air Resistance
- Additional cloth properties:
  - Stretch Stiffness
  - Bend Stiffness
- Additional soft body properties:
  - Geometry based (uses the actual triangles in the model), and cube lattice based (simulates a series of connected soft cubes which may themselves be used to drive deformation of an underlying 3D model)
  - Variable stiffness parameter.
- Rope properties:
  - Rope is a linked chain of vertices, with a user specified thickness.
  - 2 Dynamics models available: constraint based, and spring based (faster but springier)
  - Variable stiffness.
  - Variable bend stiffness.

Deformable body constraints:

Constraints allow you to connect cloth, rope and soft elements to each other and to rigid bodies in the scene. The system of constraints provided is identically applicable to soft bodies, cloth and rope. “Vertices” below means vertices in soft bodies, **or** cloth, **or** rope.

- Rigid Body to Deformable Constraints:
  - Constraints from vertices to world space positions or to user specified transforms allowing simple integration with bones systems.
  - Constraints from vertices to rigid bodies, with optional force feedback (i.e. you can allow the deformable bodies to affect the rigid bodies or vice versa).
  - Dynamic addition and removal of constraints.
- Deformable to Deformable Constraints
  - All permissible combinations supported: Rope-Rope, Rope-Cloth, Rope-Soft Body, Cloth-Cloth, Cloth-Soft Body, Soft Body-Soft Body
  - Dynamic addition and removal of constraints allows for breaking ropes, pre-scripted tearing of cloth, etc.
  - Constraints can be combined e.g. many vertices from multiple ropes, soft bodies or cloths can be constrained together
  - Deformable to Deformable and Rigid Body to Deformable constraints can be combined together.

Deflectors:

Cloth, soft bodies, and rope do not get added to the Havok collision detector. Instead, a specialized collision detection system is provided to allow very fast and accurate collision detection with specialized objects.

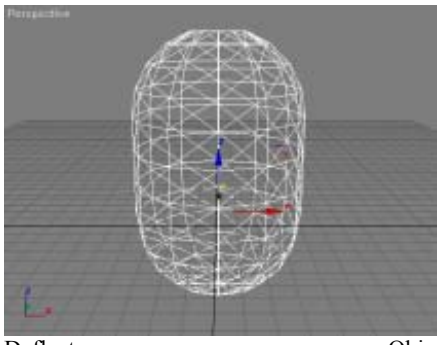
The focus of the cloth / soft body, and rope collision detection solution, is for the creation of deformable character dynamics.

Cloth vertices may be setup in advance to be deflected by any number of deflectors. In addition, deflectors can be used dynamically (activated by callback functions from the collision detector) to provide some interaction between external objects and deformable bodies. Deflector shapes provided include:

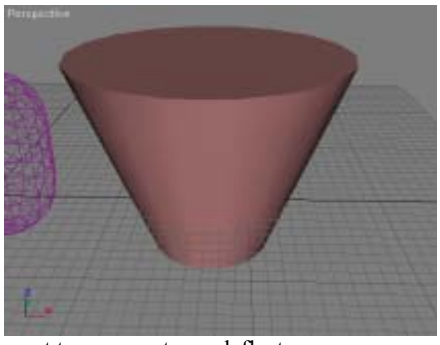
- Plane deflectors
- Cylinder deflectors
- (Almost) Arbitrary geometric deflectors = capsule height field

This last deflector type might need a little more explaining. To create deflectors for objects that are not flat (where a plane is fine) or cylindrical (usually OK for most limbs of characters) you can use a capsule height field, which is fitted to the shape of the object you want to represent. Remember, this new deformable technology does not have collision detection with arbitrary objects so you have to do a bit of work to achieve the same result. Recall that height fields are a pretty efficient way to do collision detection (that's why they're so popular on PS2 in particular). What we've provided here is a height field in 3D – one that is wrapped around a cylinder capped with 2 hemispheres (a capsule). We take the rigid body we want to represent and generate a height field on the capsule to closely approximate the rigid body. If the rigid body is convex then we can always generate a good representation. If is not convex, we will end up with a convex deflector regardless (sort of like using a convex collision geometry but a non-convex display geometry). This is the main limitation of the deflector approach.

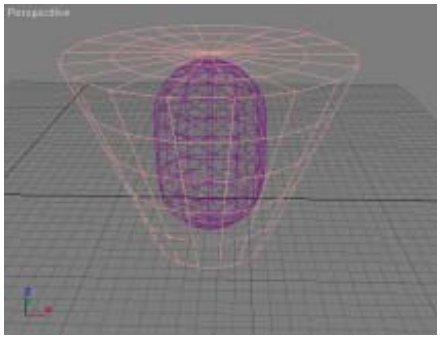
This process is shown below using screen shots from the max exporter:



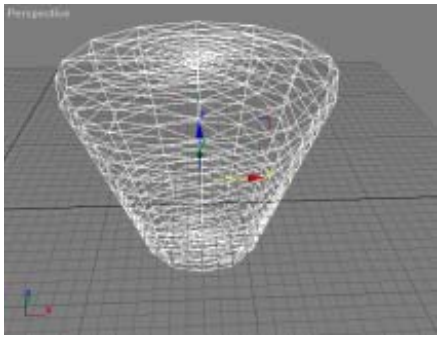
Deflector



Object we want to represent as a deflector



Object and deflector superimposed



Deflector mapped to the object

In addition a volumetric constraints system is provided, based on deflector positions, to allow easy control of the behavior of a cloth, soft body or rope interacting with a deflector. Vertices may be constrained to lie within a deflector for example (maybe to prevent some part of the cloth from deviating too far from its initial position). In this way the user can create a wide variety of real time clothing solutions, and deformable character dynamics.

8 Actions & Controllers

We provide an extensible architecture to allow game developers to add their own game specific dynamics functionality to the Havok system. The main interface is via our action framework. Actions are functions or modules called during each simulation step that have access to the entities in the simulation and can apply forces, torques and other direct manipulations of objects. A set of actions is provided with the SDK with source to illustrate how to go about creating your own.

Feature Benefits:

Feature	Description	Benefit
Action Interface	An interface to the physics SDK to allow easy extensions of the functionality and create game specific physics components.	This interface gives you access to the core of the Havok simulation and allows full control over all the entities in the system.
Z-Order Actions	A special class of actions that give you direct kinematic control over objects being simulation. This allows you to directly set positions and orientations.	Actions that alter the positions and orientations of objects can cause interpenetration problems. The Z-order action interface is designed to get around this by taking such changes into account and thus allow easy integration of key framed animation into the physics simulation.
Character Controller	A special purpose action that provides a rich set of control paradigms for moving a character around a 3d world while interacting physically with the world and the objects contained within.	This makes it very easy to get character-based games up and running using the Havok system. The controller is designed to be extended to suit your specific game play needs.
Source available	Source code is provided for all the non-core actions.	You get to see in detail how many of our actions are created giving you a great base upon which to build your own game specific actions.

The complete list of actions (with source) provided includes:

- Character Controller: a basic character controller with walk, jump run and climb actions.
- Constant Force Action: apply a constant force at each time step. This is a tutorial action.
- Magnetic Action: turns any object into a magnet.
- Motor Action: apply a motor to an object with controllable torque and gain.
- Self-Orientator: an action that uses an angular dashpot to flip an overturned object. Useful for righting flipped cars.
- Simple Drive Controller: a basic vehicle controller.
- Simple Rigid Wind Action: a wind effect that applies to rigid bodies.
- Simple Soft Wind Action: a wind effect that applies to deformable objects.

9 Events & Callbacks

The primary mechanism for interfacing the physics system with game logic and AI controllers is through our event and callback system. This is a comprehensive Java-like event delegator, event pipe, event filter and event dispatcher system to give full control over the raising and handling of events.

Feature Benefits:

Feature	Description	Benefit
Collision Events	Raised in response to collisions between certain objects in the system. The events may be filtered according to criteria including object types, collision strengths and frequency.	Collisions form one of the core events that a game will need to monitor for game specific logic (i.e. character picks up a key, or car crashes into a wall). By filtering events we give control over the strength of the collision required to create a certain effect (e.g. the car is destroyed only when it hits the wall hard enough).
Sensor Event	Many parameters of physical objects may be monitored and an event is raised if the sensor value falls within a user specified range of values (e.g. raise an event when the velocity of an object exceeds a specific value).	This lets the physics system monitor objects in the system and only inform the game logic when a certain state has been reached
Interpenetration Events	An event can be triggered when two objects interpenetrate to allow an application specific fix to occur.	Interpenetration usually occurs when an object has been placed into an invalid state by game logic code or an AI routine. By raising an event, the game can respond intelligently, taking appropriate actions to restore the objects to a valid state.
Phantom Object Events	Events may be triggered in response to an object colliding with or leaving a phantom object.	Using this event system, it is very easy to create triggers for game behaviors. For example, you can put a phantom object in a doorway and an event will be raised whenever any object tries to get through the door.
Callbacks	Events can be read directly off event pipes or alternatively callback functions can be associated with each event. These functions will be called automatically by the Havok system when the event is raised.	Callbacks provide a natural and simple way to associate game logic functions with Havok events. With callbacks there is no need to continually check event pipes through code.
Event Filters	Events may be filtered (as in sensor events) by ranges of values but also by frequency. You can specify, for example, that you only want one collision event per second from a set of rigid bodies.	You have full control over the numbers of events being generated. In situations like rock falls you can potentially have many, many events. Having a filter allows you to prevent the event pipes from filling up with unnecessary detail.

Sensor events:

The following parameters of rigid bodies many be monitored by a sensor and an event raised if the value falls within a user supplied range of values:

- Linear Velocity
- Angular Velocity
- Force experienced by the rigid body
- Torque experienced by the rigid body

10 Toolkit Layer

A toolkit layer is provided on top of the core physics API to allow easy access to many of the features of the SDK without needing to dig deep into the workings of the core. With the toolkit, you can inject physics functionality quickly and easily into your game.

Feature Benefits:

Feature	Description	Benefit
Toolkit functions	A suite of about 20 functions is provided to allow very rapid scene construction and manipulation without needing detailed information about how the core of Havok works.	Get up to speed really quickly, creating rigid bodies and entire scenes and experiment with the physics without necessarily becoming an expert. This allows you to evaluate the system rapidly and only delve deeper into the core Havok APIs when you want to have more flexibility or generality.
TK File Utility	A simple utility demonstrating reading geometry from an ASCII file format and creating the corresponding rigid bodies.	You can use this to develop your own streaming utilities and to see how core geometry in Havok is created and handled.
Export Toolkit	A toolkit to allow reading and extracting information from a HKE file exported from a Havok exporter.	This toolkit allows you to very easily read in entire levels of individual objects from a HKE file with little coding required. Very quickly you will be able to get an entire physically enabled scene up and running with your artists designing the physics in an industry standard modeler.

11 Platform Support & Renderer Integration

The first thing any game developer needs to do when using a physics engine is to interface the physics system to the existing 3D graphics / rendering solution. This should be a straightforward task, but because of the particular features of each renderer and the various platforms (particularly consoles) to be supported Havok has put a lot of effort into making the integration an easy one and to open up the core of the engine so that game specific optimizations for each platform may be implemented.

Feature Benefits:

Feature	Description	Benefit
Multi Platform	Havok is available on PS2, PC, Xbox and GameCube. We support all the major compilers on each platform and maintain up to date builds for the latest hardware updates from the platform manufacturers.	Games developed with Havok on one platform are easy to port to all other platforms supported by Havok. You are guaranteed that your physics code will compile and run across all the platforms Havok support. In some cases (e.g. moving from PC to PS2), you may want to optimize your code further due to the constraints of the platform. Havok provides you with the flexibility you need to further tune your game's performance.
DirectX8 Display library	A sample display library implemented in DX8 is provided for Win32 platforms.	This allows you to examine how we have interfaced the physics with an industry standard graphics API. You can use this information to develop your own interface to the physics system.
hkBase source	Full source for the Havok base class including memory manager and timer classes is provided allowing the implementation to be replaced by application specific routines optimized for your context	Particularly in the case of the console platforms, where organization and management of memory is critical for game performance, having full game control over the hkBase class gives the game developer the flexibility needed to employ game specific optimizations.
Active object lists	Active lists of objects are maintain internally by Havok and exposed to the rendering solution. The renderer then simply repositions those objects that have moved since the last frame.	This allows the 3D engine to streamline the updates to objects – inactive objects will not have moved since the previous frame and therefore do not need to be repositioned. This may have significant performance improvements for 3D systems that employ caching.

12 Debugging / Profiling

Havok has not been designed as a black box system. Though we need to protect some of our core IP by not supplying source, we have gone to great lengths to remove the need for source to allow you to effectively debug and test your applications.

Feature Benefits:

Feature	Description	Benefit
Visual Debugger	A graphical interface for remotely monitoring your simulation, including collision geometries and bounding boxes. Source for the Visual Debugger is shipped with the product. See Tools section for more details.	The Visual Debugger provides a visualization of the position and orientation of simulated objects. It is very useful for debugging any rendering of simulated worlds. Source code is shipped with the Visual Debugger, making it easy to configure and integrate into your own tool chain.
Debug Build	A debug build of the Havok SDK is provided on all platforms. The debug build provides many additional ASSERTS and a full call-stack to allow you to pin point where problems are occurring.	The debug build allows you to pin point the source of your problems which will either help you solve the problem or will speed up the turn around time on a fix from Havok's developer relations team.
Timing Stats	The hkBase class provides a bank of timers that may be used to time different parts of the core to examine the relative CPU usage of these different parts.	The timers allow you to identify the CPU load per module (e.g. collision detection, collision response, actions etc.) so that you can identify bottlenecks in the system and concentrate your optimization efforts on those parts that are slowest.
Source for many systems provided	Source code has been provided for a large number of non-core components.	Many of the game play and architectural level modules can be replaced entirely with user-supplied code if necessary.
Memory profiler	Our sample memory manager (provided with source through the hkBase class) tracks memory usage and can provide a full memory trace at any time during the execution of the application.	Track memory allocation and de-allocation to help pin point the source of memory leaks or heavy memory usage.

13 Havok: All Tooled Up

Havok provides a comprehensive set of tools for creating, tuning and profiling your physical content:

- Havok SDK Exporters – set up and tweak your dynamics scenes within a familiar environment.
- Car Tuning Tool – assign and tweak over 100 vehicle parameters in real-time, while previewing your game on your target platform
- Visual Debugger – analyze your scene while running your game on your target platform: identify synchronization issues and profile your game's performance.
- Tools to help you build your own – Havok also provide the tools necessary to incorporate our tools into your own toolchain.

Create your content in a familiar environment

From within your favorite modeler and without writing any code, the **Havok SDK Exporters** enables you to assign physical properties to elements in your scenes and levels exactly as you would assign color, textures and game-specific attributes. You can even preview the behavior of your work real-time within the modeler. The result is speedier development.

Features and benefits:

- **Work in a familiar environment**
  - The Havok SDK Exporters work from within 3ds max and Maya so that you can take advantage of editing tools and workflow provided by your favorite modeling tool.
- **Allows you to create complex physical systems quickly and easily**
  - Easily assign physical parameters to rigid bodies.
  - It's not just the easy-to-handle mass or friction properties that can be applied within the modeling tool. Complex systems of constraints such as articulated bodies and hinges can be created.
  - Easily create, tweak and test character joints.
  - Simple geometries act as proxies for more complex geometries, and bodies can be grouped to form more difficult ones.
  - Enable and disable inter-object collisions, or create collision layer groups for more fine-tuned control of the physical scene.
  - You can update the state of your scene in the modeler after a preview.

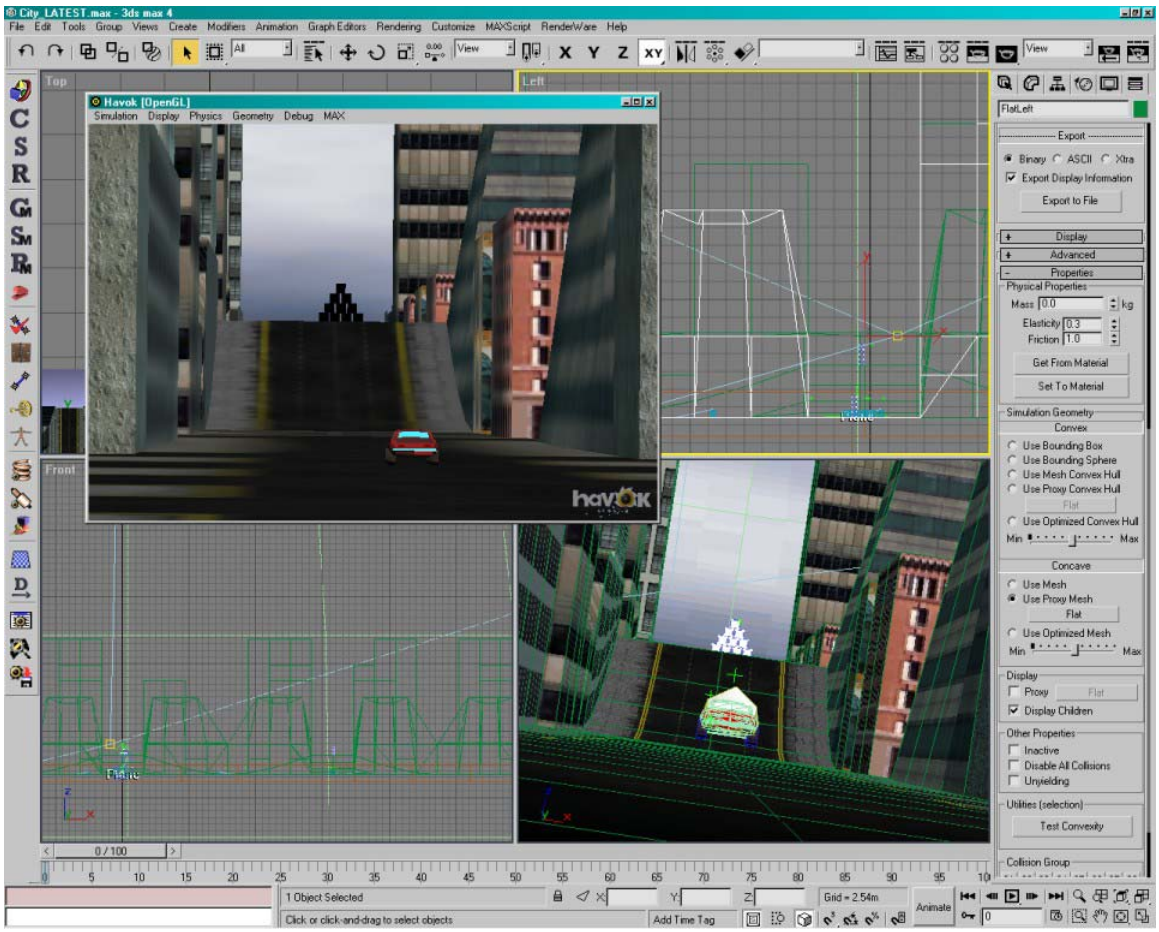


Figure 1 Havok SDK Exporter for Max: setting up a scene in a familiar environment

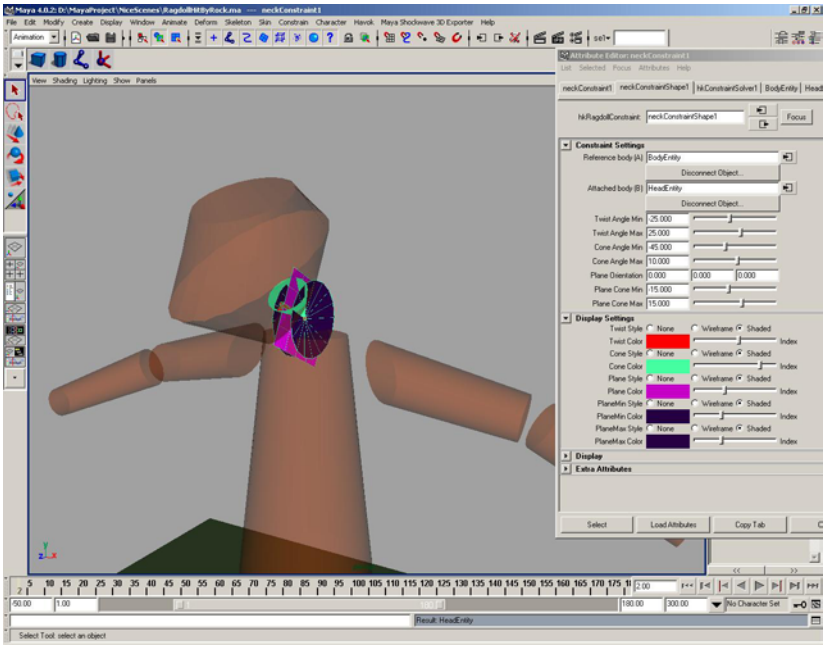


Figure 2 Setting up rag dolls in the Havok SDK Exporter for Maya

- **Reduces the dependency between designers and programmers**
  - Physics contend created in max/Maya can be previewed in real-time using our preview display (OpenGL or DirectX).
  - Real-time preview within the modeling tool allows artists and designers to quickly prototype physical interactions without having to wait for the game engine to be up and running.
  - In preview mode you can examine the geometry of the objects, play and pause the simulation and explore the scene with a user controllable camera and a mouse-picking interface.
- **Exporters provide an easy path to the game engine**
  - Once exported, files are imported by a game with minimum amount of code and, once imported, all objects and their parameters remain fully accessible via the names assigned in the modeler. This makes the task of writing customer controllers or systems acting upon the physical objects much easier.
- **Allows you to debug and profile your scene**
  - With the realtime preview you can inspect and debug the physics scene interactively.
  - Analyze your physical world to find possible inconsistencies such as interpenetrating objects or values that are out of range.
  - Optimize performance by assigning rigid bodies to collision group layers.
  - Determine the best collision geometry proxy to use and optimize the proxy automatically (mesh reduction) or manually (define your own geometry).
  - Various display modes are supported (solid, wireframe, physics geometry only etc.) to give you full control over the playback of the physical world. Artists and designers can therefore debug much of the physics functionality before turning the scene over to the game programmers.

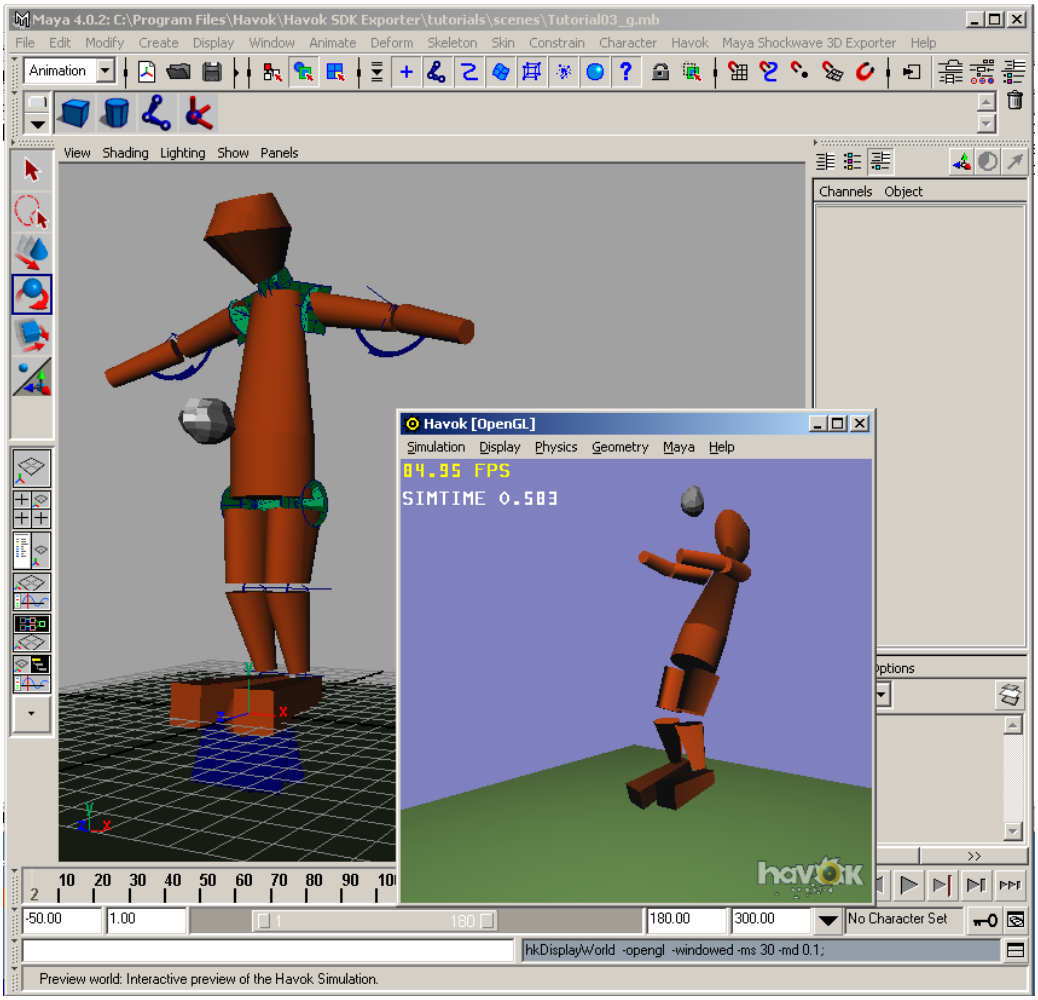


Figure 3 Real time preview in Maya

Tune, debug and profile your game using on-target tools

Once all your physical content has been imported into your game engine, developers can use Havok's virtual tuning and profiling tools to tweak and analyze the game on the target platform. **Havok's Car Tuning Tool** allows you give your game vehicles a virtual overhaul, while viewing the results on-target in real-time. The **Havok Visual Debugger** provides a graphical interface for remotely monitoring your game.

Figure 4 On-target real-time preview of vehicle game

Features and benefits:

- **Tweak a few slide bars to change your vehicle behavior from a Formula One racer to a cartoon truck**
  - A default vehicle is constructed from more than 50 unique parameter values. These are placed into some ten categories that define the car, such as steering, suspension and wheels.
  - Tweak the default values to design your unique vehicle handling.
- **See the vehicle behavior change in real-time on your target platform**
  - Changing a parameter value in code and recompiling the executable is obviously time-consuming and bothersome. But with our tuning tool you can dynamically load the parameter values from an external source, or synchronize using a real-time connection.
  - Run a car simulation on the PlayStation2 and tune it using the tuning tool running on a PC.

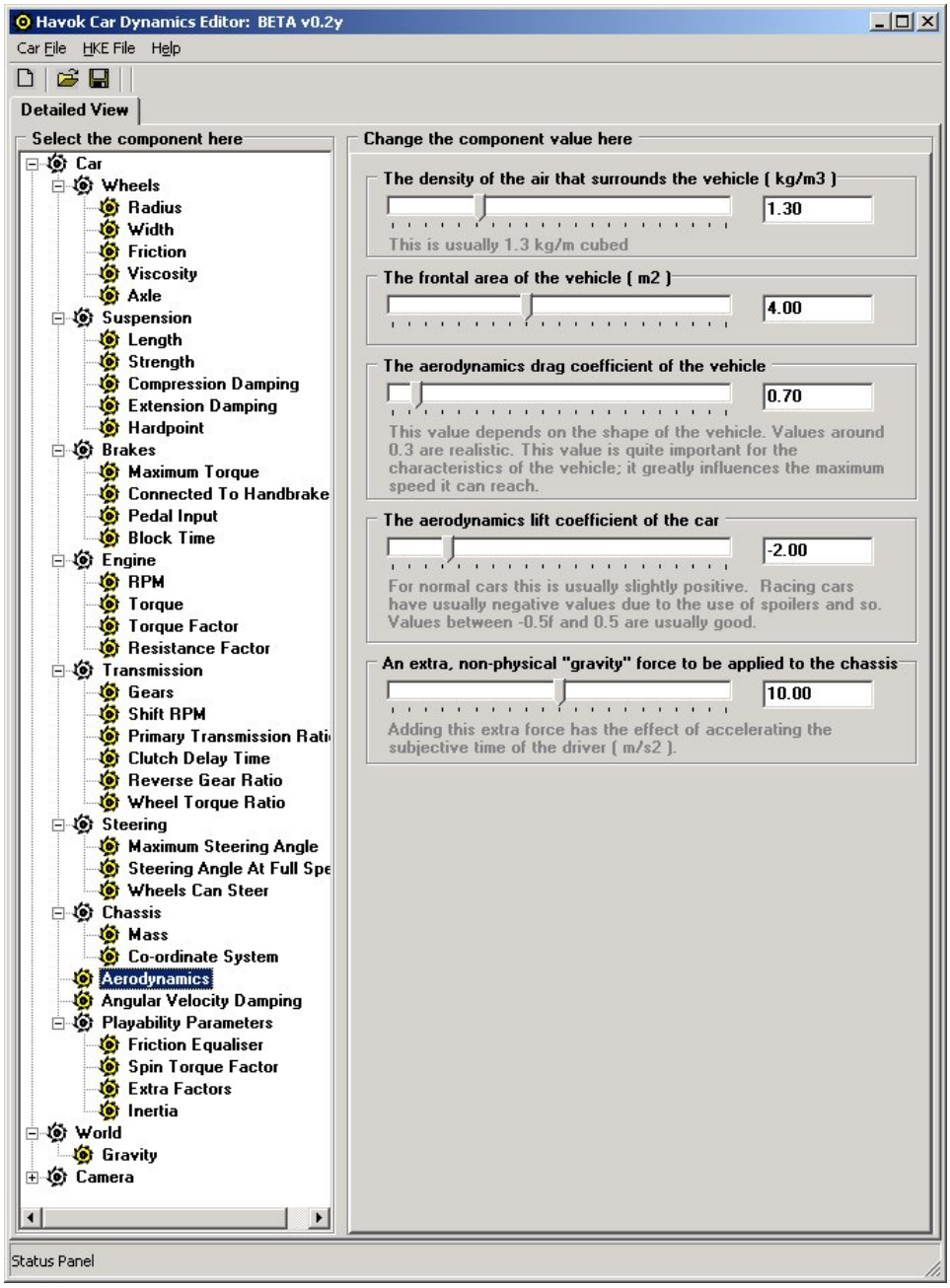


Figure 5 Tweaking over 100 vehicle parameters in real-time

- **Easily check for any inconsistencies between graphical and simulation worlds**
  - Visualize collision geometries and bounding boxes to check position and orientations of simulated objects, while playing your game in real-time on target platform.

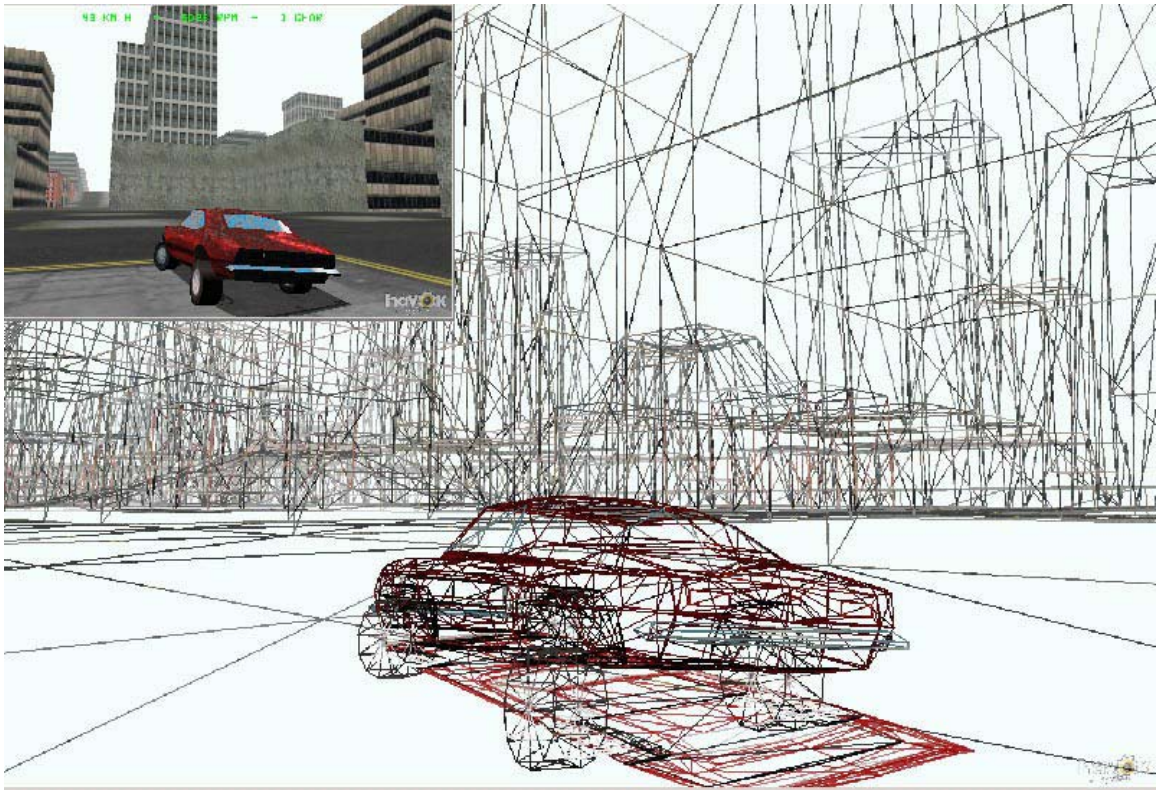


Figure 6 Visualizing collision geometries and bounding boxes in real-time, while demo is running on target platform.

- **Check for rogue set-ups**
  - Timings and statistics displayed in the Visual Debugger allow you to check your set-up and identify areas where you are getting less than optimal performance.

Bend them, shape them, any way you want them

All the Havok tools are designed so that they can be easily integrated into your tool-chain. We provide the means by which you can integrate our export functionality into your own exporter, writing Havok date into a game specific file format.

Source is provided for both the Car Tuning Tool and Visual Debugger, so you can extend both tools and easily integrate the functionality into your own toolchain.

14 Demos, Demo Framework and Documentation

Havok is provided with a comprehensive set of demos, all provided within a demo framework that abstracts the simulation from the display and provides for cross-platform support (abstracts mouse, keyboards and controllers). There is a wealth of documentation available also covering all aspects of the physics system at a user level and an online reference manual.

Feature Benefits:

Feature	Description	Benefit
Reference Manual	A complete reference manual in CHM format is provided with documentation on all Havok classes and functions. Full hyper linking is provided with class hierarchy diagrams, a search function and a complete index tool.	Quickly get information on any Havok function or class. Perform fast searches and cross-reference your searches through the hyper links.
User Manuals	A set of user manuals covering all aspects of the SDK, the API and tools is provided	A wealth of reading material should ensure you are never in the dark about any aspect of Havok technology. Code samples are provided for cutting and pasting into your applications and many manuals cross reference with the code samples provided in the demos so that you can always find source associated with the feature you are looking for information on.
Demos	Numerous demos are provided with full source covering all aspects of the Havok engine.	Get source code examples for all aspects of the engine so you can quickly cut and paste into your own application and build on the examples provided.
Demo Framework	A demo framework is provided which abstracts the differences between the various hardware platforms supported by Havok and also separates the display and simulation aspects of each demo.	This abstraction leaves the physics source in one place so that it is easy to read and understand without being confused by controller handlers and specific display information.

List of Manuals:

Note: all manuals are in Adobe PDF format and are fully illustrated in color.

- *Actions – Poltergeist and Possession*: describes the action interface and illustrates how to create your own actions to extend the physics system using a poltergeist action as an example. This is accompanied by a corresponding demo with the action fully implemented.

- *Collision Detection*: a complete overview of the Havok collision detection system, discussing geometry formats, narrow and broad phase collision detection systems and how to interface with the collision detector yourself.
- *Constraints User Guide*: an overview of the complete Havok constraints system and associated parameters.
- *Convex Hull Generator Utility*: a description of the simple utility supplied for max that allows you to break complex objects into simpler convex sub-parts.
- *Deformable Bodies User Guide*: a complete overview of deformable bodies, cloth, soft and rope and associated constraint systems.
- *Events – Pinball Wizard*: a tutorial on using the Havok events systems covering all events by creating a simple pinball-like application.
- *Havok Base*: an overview of the hkBase case, for which source is available.
- *Havok SDK User Guide*: an introduction to the Havok physics system. You should read this manual before doing anything else. It explains the architecture and philosophy of the Havok SDK and gives an overview to all the remaining documentation.
- *Quickstart Guide*: a guide to compiling and running the demos and getting your first application up and running using Havok.
- *Vehicle SDK Programming Guide*: a complete description of the architecture and class hierarchy of the vehicle SDK.
- *Vehicle SDK User Guide*: a comprehensive overview of the vehicle dynamics SDK with a description of all the parameters and modules and hints on how to get the performance you require from the SDK by hacking the physics.
- *Havok Vehicle SDK Workflow*: a guide to the car tuning tool and how to integrate it into your own vehicle design workflow.

List of Demos

The following list gives all demos shipping with Havok 1.7. For each demo, the features demonstrated are listed.

Demo	Features
Fast Subspace	Lots of objects colliding, both in gravity-free and with-gravity situations. Each of the objects is actually simulated as a sphere.
Stacking Boxes	Stable stacking with lots of boxes on top of each other. This highlights good collision detection and scalable collision response. The boxes are simulated with bevelled edges.
CharacterCloth	Shows how to create a cloth object and a deflector. The demo illustrates cloth collision with a complex shape with small CPU overhead.
CharacterCloth Controller	Demonstrates how to handle a character animated using bones and our fast cloth technology. Deflectors have been setup for each bone (i.e. limb) and these move when the limbs move in the keyframed animation, so that the cloth animates naturally with the character.
ChildsPlay	Shows stable collisions and collision response. Long thin objects and a spinning top, that is initially spinning, exhibit the classic behavior of the kid's toy.
CityCar	Havok's vehicle physics in action. We have a large landscape with Driver style gameplay on the car (US style suspension – large heavy chassis, lots of body roll etc.).

Demo	Features
ClothCollision	A complex object collides with lots of high-density cloth patches. This illustrates the speed of the cloth simulation. Note that to achieve this high performance level, the cloth does not avoid self-penetration – an acceptable compromise in most cases.
EventTest	Demonstrates (mostly for coders) the event system in Havok. Events are raised when the cube collides with the plane and also when the velocity of the cube is between 5 and 10 m/s (but filtered so that only 1 such event per second will get handled). This is not a great standalone demo – most value is gained from looking at the code.
HingeConstraint	Shows the fast hinge constraint in Havok. Each of the cubes is attached to the next via a hinge joint (collisions between every neighboring cube pairs are disabled).
hkMenuGame (blank)	A blank demo application just showing, through code, how to setup a demo yourself.
LP2PConstraint	Limited point-to-point constraint demo shows how to setup and use point-to-point constraints. Joints limits limit the permitted angle between the constrained bodies giving more control over the behavior of the joints.
Necklace	This scene has been created in max and exported as a HKE file. Each of the links of the necklace is represented as simplified proxy geometries (turn on sim-edges to see these) attached together with point-to-point constraints. It's easy to get the necklace into an impossible situation because we haven't limited how each link can twist relative to the ones attached, but the simulation is fast and convincing otherwise.
P2PConstraint	A long chain (30) of cubes attached with point-to-point constraints. Note how the end cubes get snapped around a lot but don't drift or seem springy – this is what people will be looking for. Also, compare this to the limited P2P version: you can see that these cubes spins lots more as there are no angular limits here.
PhantomObjects	This addition to collision detection allows you to specify an object as a region rather than solid, with collision events raised accordingly. In this demo pick up the small cube and move it through the walls – an event is raised on entering and leaving the walls.
PileUp	Shows the robustness of the collision detection and response: lots of different shaped objects (all convex) congregate around a narrow opening at the base of the "chute". This is often a difficult case for collision detectors to handle.
Pinball	A tutorial for Havok events and sensors. Collision events with the middle cylinders cause impulses to be applied to the object, so that it bounces off rapidly. A phantom object at the base (invisible) allows us to know when the falling object has reached the bottom. Finally a velocity sensor is used to add to the score based on the speed the ball travels – higher marks for higher speeds.
Poltergeist	Another tutorial showing how to create your own special purpose "action". In this case the action checks if an object is inside a cone region in the center of the room and applies a tornado like force if it is.
RagDoll	This demo uses the ragdoll constraints (one for each limb connection) to create a ragdoll that may be picked up and chucked around or "kicked" using the keyboard.
Rope	Shows rope collisions and simulation with a head. It also shows how to create a deflector from geometry through code. The ropes are spring based as apposed to constraints so exhibit a little more springiness but with the advantage of low CPU overhead. If you want stiffer ropes, use the constrained variety.

Demo	Features
SoftBody	A cube with wobbly soft body antennae can be controlled showing how the soft bodies collide against the objects in the scene (which have had simple deflectors associated with them through code). You can also pull the soft bodies with the mouse.
SSpringConstraint	Stiff springs demo – compare this with the P2Pconstraints demo. They do essentially the same thing, but stiff springs are faster to compute but exhibit a little more springiness which you can see near the top when the chain reaches full extension – the cubes at the top are pulled away from each other slightly. This is not so noticeable on shorter chains.
Toothpicks	A demo of collision handling for objects that are typically difficult to handle i.e. long thin ones. You can select between normal long and thin and even longer and thin so see the effect.
Truck	Another example of the vehicle SDK, this time with simple AI opponents. These guys follow a set of way points on the track, but all the behavior of the trucks (skidding out, under steering etc.) are all down to the vehicle SDK. Power sliding and tight suspension is a feature of the truck (in comparison to the Driver style loose suspension in the city car demo).
WindAndCloth	A flag flapping in the breeze just showing the nice behavior of the cloth coupled with a wind action. The cloth can be picked, but this doesn't look very nice. There are 1100 triangles in this piece of cloth.

15 Source Code

The source code that is shipped with the SDK is listed below.

- Events
- Base Library (hkBase)
- Toolkit Library
- Vehicle SDK gameplay modules
- Particle System (available on request; mail info@havok.com for more details)
- Water Module (available on request; mail info@havok.com for more details)
- Renderer Integration Toolkits
- Demos
- Car tuning tool and Convex object tools.
- Core Vehicle physics.

Integration source code is provided for the exporters.

Source is NOT shipped for:

- Core actions (including constraints)
- Collision Detection
- Collision Resolution
- Entities
- World
- Math Library
- Geometry Lib
- Export Lib

16 Contact Details

European & UK Offices

1 Farnham Road,
Guildford,
Surrey,
GU2 4RG,
Tel: +44 1483 549 287
Fax: +44 1483 549 100

North American Offices

510 Veterans Blvd,
Redwood City,
CA 94063,
USA
Tel: +1 650 322 2332
Fax: +1 650 322 2240

Global Headquarters

7 Westland Court,
Cumberland Street,
Dublin 2,
Ireland
Tel: +353 1 677 8705
Fax: +353 1 676 7094

Further Information: info@havok.com
Support: support@havok.com
World Wide Web: www.havok.com