Example Programs for MathEngine Toolkits

•

Overview

This document describes the example programs distributed with the MathEngine Toolkits.

The example programs are short programs designed to show you how to write programs using the MathEngine Toolkits, particularly the MathEngine Dynamics Toolkit and the MathEngine Collision Toolkit. Source code is therefore included.

Example programs are in the `examples` folder.

Some demonstration programs (demos) are also example programs. Demos are usually longer programs designed to show the power of the MathEngine Toolkits. The demonstration programs that we describe in this document include the source code.

Demonstration programs are in the `demos` folder. See *Demonstration Programs for MathEngine Toolkits* for information about all demos.

•

Standard User Interface

Most example programs and demonstration programs use the MathEngine Viewer both for rendering and to provide a standard user interface for viewing the rendered scene and controlling the simulation. See the *MathEngine Viewer Developer's Guide* for more information about the MathEngine Viewer*.*

MathEngine Viewer Commands

For any program that uses the MathEngine Viewer, you can usually manipulate the rendered scene by using the standard viewer commands below:

Control Keys

Control Keys	Action
cursors	move camera
ctrl+cursors	move light
F1	Application-specific Help
F2	change shade model
F3	toggle texture
F4	toggle axis display
F5	toggle stats display
F6	toggle text
F7	toggle camera info
F8	toggle display lists
F9	toggle fullscreen
F10	toggle pause
F11	toggle mouse manipulation
F12	toggle renderer help
<Esc>	quit

Mouse Controls

Mouse Controls	Action
drag in center	orbit camera
drag near left edge	pan camera up/down
drag near bottom edge	pan camera left/right
drag near right edge	pan camera towards/away
drag near top edge	zoom camera in/out
SHIFT + pick graphic	translate object left/right and up/down
CTRL + pick graphic	translate object towards/away
ALT + pick graphic	rotate object

Notes for Windows and UNIX

- The standard commands are displayed automatically by the MathEngine Viewer, usually at the beginning of an application’s execution.

-

- F1 displays Help provided by the application program, usually for keyboard commands such as <Space> or <Enter>.
- F12 provides Help for the MathEngine Viewer itself.
- F1 and F12 stop the simulation while they display the Help. Press F1 or F12 again to resume the simulation.
- Not all programs support the pick graphics commands.
- Many demonstration and sample programs involve throwing balls from the camera towards objects in the scene. You can manipulate the camera in order to aim before you throw.
- Some programs override the default Viewer commands. Press F1 to check for any overrides and to receive application-specific Help.
- Clicking the right-mouse button on a rendered scene provides a menu of commands.

Notes for Sony Playstation2

The Sony PlayStation2 is normally used with a control pad, not a mouse and keyboard. Most of the controls listed above are not supported. You can, however, rotate the camera by using the left and right buttons on the control pad.

Searching the Code for User Interface Details

For the demo programs and example programs that are distributed with MathEngine Toolkits: if the program uses the MathEngine Viewer, you can see how application-specific commands are implemented by editing the source file of the program, and searching for `RUseKey` or `RUsePadKey`.

Here, for example, is how `cd_topple` implements its commands:

```
#ifndef PS2
     RUseKey('s',  stepEvolve);
     RUseKey('a',  toggleAutoEvolve);
     RUseKey(' ',  shoot);
     RUseKey('k',  killAll);
     RUseKey('w',  wakeAll);
     RUseKey('f',  toggleFriction);
     RUseKey('\r', reset);
  #else

     RUsePadKey( PADRup, stepEvolve);
     RUsePadKey( PADcircle,   shoot);
     RUsePadKey( PADRleft,    killAll);
     RUsePadKey( PADRdown,    wakeAll);
     RUsePadKey( PADtriangle, toggleFriction);
     RUsePadKey( PADsquare,   reset);

  #endif
```

drop

Toolkits Demonstrated: Dynamics

Simulates a single falling ball.

This is the simplest example using the Kea Solver of the Dynamics Toolkit. The example demonstrates a single body with only gravity applied.

Usage

- <Enter> resets the simulation.

•

bounce

Toolkits Demonstrated: Dynamics

Shows a ball dropping on a plane and bouncing.

This example demonstrates the use of contacts with the Dynamics Toolkit, but without using the Collision Toolkit.

Additional balls can be thrown by pressing the spacebar. You can move the camera to aim before throwing.

Usage

- <Space> drops the ball.
- <Enter> resets the simulation.

cd_ball

Toolkits Demonstrated: Collision

Uses Collision Toolkit alone, *without* Dynamics Toolkit.

The scenario: a ball is thrown at a wall, bounces off the wall, bounces off the floor, and then exits the scene.

Note: There is no gravity in this simulation.

Usage

- <Space> throws the ball. You can press <Space> as often as you wish.
- <Enter> resets the simulation.

Discussion

The motion is controlled by a very simple physics algorithm in `cd_ball.c` itself.

This program demonstrates how to use the Collision Toolkit to perform some basic tasks:

- Obtaining collision events.
- Identifying them (“ball-wall” collision vs. “ball-floor” collision).
- Calling the intersection function.
- Reading contact data structures.

The code for the motion algorithm shows you how to extract information from the contact data structures and how to use this information to produce appropriate changes in the motion of the rendered models - in this case both bouncing off the wall and floor.

The program also show how the new positions computed from the motion algorithm are used to update the coordinate systems for both the collision model and the rendered model.

•

cd_ball2

Toolkits Demonstrated: Dynamics

Reproduces the behaviour of cd_ball, only now the Dynamics Toolkit is being used to control the motion.

Usage

- <Space> throws the ball. You can press <Space> as often as you wish.
- <Enter> resets the simulation.

Discussion

There is no motion code anywhere in the program. All motion is controlled by calls to the Dynamics Toolkit.

The Dynamics Toolkit bridge, which is a component of the Collision Toolkit, handles all communication and synchronisation between the Collision Toolkit and the Dynamics Toolkit.

The only responsibility of the application programmer is to specify the geometrical and dynamical properties of the simulation (via the Collision Toolkit and the Dynamics Toolkit interfaces, respectively).

cd_ball3

Shows how small changes to `cd_ball2` achieves dramatically different behavior.

The first example is pre-compiled and ready to execute. The variations below are gnerated by performing small macro replacements directly in the program code. You need only perform these simple edits and recompile to obtain an executable.

Changing a Ball to a Box

By changing one line of code, the bouncing ball becomes a spinning-and-tumbling box undergoing realistic collisions with the wall and floor.

Adding a Bar

Now a second projectile, a long "2x4" bar, is added to the scene. The bar and the box spin, tumble and collide with the wall, the floor, and each other. This demonstrates the ability of the Collision Toolkit to handle multiple pair-combinations of collisions correctly, efficiently and automatically.

The complexity of program code increases only linearly with each additional projectile (just duplicate the code that created the first "ball") although the number of collision interactions that are being handled increases quadratically.

Changing the Dynamics

The above two variations demonstrate the ease with which geometrical properties can be modified to produce dramatically different results. This involves simple changes to the code that uses the Collision Toolkit.

The following two variations also begin with the code of `ball_hits_wall2.c`, but produce dramatically different results by applying small changes to the *dynamical* specification. This time, the simple changes involve code that uses the Dynamics Toolkit.

Demolition Ball

A single-line code change replaces ball-bounces-off-ball with ball-*knocks-over*-wall! Another single line "turns on" gravity, making the wall fall down instead of floating away.

The wall traps the ball, demonstrating a stable coming-to-rest transition, involving the collision-geometry configuration of box (wall) on sphere (ball), sphere on z-plane (floor), and box on z-plane.

Door Swing

You can stop the wall from falling over by fixing it the floor with a hinge joint. Now when the heavy ball hits, the wall acts like a door, swinging around. Since there are no other contraints, it swings all the way around (360 degrees), coming back to hit the ball again.

•

cd_bridge

Toolkits Demonstrated: Dynamics and Collision

Balls are thrown at a plank bridge; the bridge responds to the shock in a realistic way.

Usage

The program uses an automated cycle to throw two balls at a time at the bridge at pre-defined intervals. It varies the parameters each time it throws.

Dynamics Discussion

The planks are attached to each other with single ball and socket joints running down the centre of the bridge.

Collision Discussion

This example illustrates how to selectively turn off collision detection for pairs of models that are close or touching, but whose geometrical interaction is being modeled by other means (in this case, by imposing a geometrical constraint into the dynamical model).

cradle

Toolkits Demonstrated: Dynamics

Demonstrates Newton's Cradle. Four pendulums hang freely under gravity.

Usage

Key	Action
1	lifts the first ball

Discussion

Ball and socket joints anchor the pendulums to the world. Pressing **1** applies a force to the first ball which sets the cradle in motion.

•

hinge

Toolkits Demonstrated: Dynamics

Demonstrates a hinge joint powered by a limited-force motor.

Usage

Key	Action
<Space>	toggles the motor on and off
l	toggles the limit on and off
s	switches the direction of the motor
z	decreases desired velocity
x	increases desired velocity
k	increases max force
m	decreases max force

Discussion

A body is connected to the world with a hinge joint. The hinge can be motorized and you can set rotational limits.

fixedpath

Toolkits Demonstrated: Dynamics

Shows how to use a fixed path joint.

A fixed path joint allows you to precisely control the position and/or rotation and linear and angular velocities of a dynamic body in such a way that it interacts properly with other dynamic objects. For example you could control the body using key-frame animation or using input from a user's mouse motion.

In this example a box is constrained to move on a circular path and slide along its long axis. The box rotation is derived from the simulation, and no constraints are applied, allowing to rotate freely.

The red sphere is used to show the position of the joint and is not a physics body. By default the joint acts on one body and the world. By `#define`-ing `TWO_BODIES` the joint can be made to involve two bodies: both the box and the ring.

•

cd_topple

Toolkits Demonstrated: Dynamics and Collision

A simple game that uses the Collision Toolkit and the Dynamics Event Manager.

Numerous objects are stacked into piles which you can knock over by throwing balls at them. Click and drag the mouse to rotate the view and take aim, then press **<Space>** to throw a ball.

Usage

Key	PlayStation 2	Action
<Space>	**Circle**	fires a ball
<Return>	**Square**	reset
a		toggles auto-evolve
f	**Triangle**	toggles friction
s		step evolve
k	**Left**	kill all objects
w	**Down**	wake all objects
F8		toggles a render-mode which shades objects a darker color if they are disabled.

Discussion

This example shows simple integration of collision with physics for numerous objects. You can set parameters for the contacts between objects. In this example we use three contact points between pairs of objects, with some friction and restitution. The Dynamics Event Manager is equally effective regardless of the number of bodies in your scene.

The rigid-body solver in the Dynamics Toolkit is very fast and stable. You can simply change settings for an object’s mass, friction, gravity, and so on to achieve the desired behaviour. No parameters need tuning to make this scene.

If you toggle friction off (**f**) while playing the game, you will see how important friction is in making a realistic simulation: without friction, the objects slide as if they were on a super-slippery ice-skating rink.

The Dynamics Event Manager of the Dynamics Toolkit allows for efficient processing by determining which objects need to be processed by the solver. Objects at rest are deactivated and not processed by the solver in the simulation. Collisions and forces will reactivate an object for processing.

Normally, objects that are at rest for a brief period become deactivated. You reactivate them by hitting them with a ball (or by causing another object to hit it).

But `cd_topple` also lets you press **k** to de-activate objects in the scene, and then re-activate them either by throwing balls, or by switching all of them them on by pressing **w**. If you de-activate the objects while they are in motion, they will “freeze” into unlikely positions.

By using the **F8** key, you can see which objects are disabled.

•

car

Toolkits Demonstrated: Dynamics

A very simple vehicle simulation with steering and suspension using the CarWheel joint.

The vehicle is controlled by dragging the mouse in the display window.

cd_ballnchain

Toolkits Demonstrated: Dynamics and Collision

Balls are thrown at a hanging chain; the chain responds to the impact in realistic manner.

Usage

The program uses an automated cycle to throw a ball at the chain at pre-defined intervals from various angles..

Key	Action
a	toggles pause
t	toggles throwing the the ball
f	increases the time step by 0.001 s (f - faster)
s	decreases the time step by 0.001 s 9s - slower)

Dynamics Discussion

The top link of the chain is fixed to the world at a specified location using a hinge joint. You can modify attributes and parameters.

Collision Discussion

The chain is made up of interlinked toruses. The collision detection is done on the inside of a torus. This is more difficult that detecting collision between convex objects

•

cd_chair

Toolkits Demonstrated: Dynamics and Collision

It demonstrates how to create a geometrical composite object (chair) using a set of primitives available in the Collision Toolkit.

keaBSJoint

Technology Demonstrated: Kea Solver

A simple pendulum, with and without direct calls to Kea Solver API

This example shows how to write an application that calls the Kea Solver directly, and contrasts it (using `ifdef`s) to an application that uses the Dynamics Toolkit's top-level API, the Dynamic Abstraction Layer.

The trade-off is between optimization of size and speed (using Kea only) vs. simplicity and speed of development (using the Dynamics Abstraction Layer).

The KEA_ONLY `ifdef` works like this:

```
ifdef KEA_ONLY
     Build using mdtkea.lib only;
     Use lots of your own well-optimized code to prepare Kea's inputs;
     Call KeaStep() (the Kea Solver) directly, with lots of parameters;
  else
     Build using mdtkea.lib and other mdt libraries;
     Use just a little code to call top-level API;
     Call mdtWorldStep() with just a few parameters;
  endif
```

The example program is a simple pendulum consisting of a cuboid connected to the world by a ball-and-socket joint.

The red sphere graphically depicts the position of the joint. There is no rigid body in the "world" that corresponds to this red sphere: the sphere is not part of the simulation.

•

pendulum

Toolkits Demonstrated: Dynamics

A pendulum made up of three balls.

This example demonstrates a complex pendulum made up of 8 bodies connected by ball and socket joints. Only three bodies are displayed.

Eventually small integration errors induce asymmetry.