Chipmunk2D - Game Physics Engine

Luthfi Mawarid

Computing and Software Department
Faculty of Engineering
McMaster University

May. 12, 2016

Overview

Purpose of the Software

- Free and reliable open-source physics library for game developers
- Allows developers to implement realistic physics in their video games
- Leads to production of higher quality games at little extra time and cost
- Fast, portable and lightweight
  - High reusability - bindings and ports are available in several other languages (e.g. Python, Java, Objective-C, C++, etc.)
  - Fast and lightweight - efficient and provides physics modelling functionalities without significantly increasing the resource cost of the games

Software Inputs

Software inputs mainly involve the initial physical properties of the rigid body (or bodies) being simulated, such as:

- Initial kinematic properties of the rigid bodies (such as initial mass, velocity, position, orientation, angular velocity, etc.)
- Surface and material properties of the bodies (such as friction, elasticity, coefficient of restitution, etc.)
- Properties of constraints (such as the length of a pin joint, positions of where the bodies are held by the constraints, the error reduction parameter, etc.)

The software will validate and ensure that the inputs satisfy physical constraints.

Examples of Input Variables

Var	Quantity	Unit
d	displacement	m
v	velocity	m/s
a	acceleration	m/s^2
C_R	coefficient of restitution	unitless
m	mass	kg
L	length	m
V	volume	m^3
k	force constant	N/m
ρ	density	kg/m^3
ϕ	orientation	rad
ω	angular velocity	rad/s
α	angular acceleration	rad/s^2
ζ	damping coefficient	unitless

Software Outputs

- From the input data, Chipmunk will determine the kinematics of rigid bodies undergoing collision/acted upon by a force over a period of time
- Outputs information such as velocity, force, angular velocity, etc. (see examples below)

Var	Quantity	Unit
p	position	m
v	velocity	m/s
F	force	N
ϕ	orientation	rad
ω	angular velocity	rad/s

System Architecture and Design

- Modularization - the library is divided (decomposed) into *modules* based on information hiding and design for change
- Each module has a specific role - usually contains a data structure implementation and a collection of related functions
- Module hierarchy - 3 top-level modules:
  - Hardware-hiding
  - Behavior-hiding
  - Software decision

System Architecture and Design

Modules are divided based on their 'secrets' (underlying algorithms and data structures in the implementation):

- Abstraction - users don't need to know implementation details
- Maintainability - makes it easier to maintain, debug and update implementation code (likely changes)

Installation

The release version provides three different build scripts:

- XCode (for Mac/iPhones)
- MSVC (Microsoft Visual Studio C++)
- CMake build script

Test Cases

System Testing

Example: Projectile motion

Input:

$$p_i = \begin{bmatrix} 50 \\ 50 \end{bmatrix} \quad v_i = \begin{bmatrix} 5 \\ 4 \end{bmatrix} \quad a = \begin{bmatrix} 0 \\ -9.8 \end{bmatrix}$$

$$t = 0.5, 1.0, 1.5, 2.0\, s$$

Expected output:

$$p_f = \begin{bmatrix} 50 + 5t \\ 50 + 4t - 4.9t^2 \end{bmatrix} \quad v_f = \begin{bmatrix} 5 \\ 5 - 9.8t \end{bmatrix} \quad a = \begin{bmatrix} 0 \\ -9.8 \end{bmatrix}$$

Test Cases

Unit Testing

Example: CPBody Module

```c
#include <assert.h>
#include "cpBody.h"

int main(void)
{
    // set rigid body with 50.0 kg mass and 20.0 kgm^2 moment of inertia (MoI)
    cpBody *body = cpBodyNew(50.0f, 20.0f);
    // these should pass
    assert(body);
    assert(cpBodyGetMass(body) == 50.0);
    assert(cpBodyGetMoment(body) == 20.0);
    assert(cpBodyGetAngle(body) == 0.0);
    assert(cpBodyGetType(body) == CP_BODY_TYPE_DYNAMIC);
    // the following will fail as quantities must be nonnegative
    // cpBodySetMass(body, -50.0f);
    // cpBodySetMoment(body, -10.0f);
    // changes body type to static
    cpBodySetType(body, CP_BODY_TYPE_STATIC);
    // these should pass
    assert(cpBodyGetMass(body) == INFINITY);
    assert(cpBodyGetMoment(body) == INFINITY);
    // frees and destroys body
    cpBodyFree(body);
    // program should exit without memory leaks
}

```

References

- S. Lembcke. *Chipmunk Game Dynamics Manual*[Web]. Available at https://chipmunk-physics.net/release/ChipmunkLatest-Docs/

The following resources are all available at the se4sc repository:

- A. Halliwushka. (2015, August 4.). *Module Guide for Open Source Game Physics Library*[PDF].
- A. Halliwushka. (2016, January 19.). *Software Requirements Specification for Chipmunk2D*[PDF].
- A. Halliwushka. (2015, June 14.). *Verification and Validation Plan for Open Source Game Physics Library*[PDF].
- S. Smith. (2015, June 4.). *Modular Design*[Presentation].