HƯỚNG DẪN SỬ DỤNG THƯ VIỆN

OPEN DYNAMICS ENGINE

Nguyễn Văn Trường

Ngày 10 tháng 05 năm 2004

Bản dịch này thuộc bản quyền của tác giải

MỤC LỤC

GIỚI THIỆU 7

1.1 Tính năng của phiên bản hiện thời 7

CÀI ĐẶT VÀ SỬ DỤNG 9

2.1 Cài đặt 9

2.1.1 Dịch và chạy ODE các ví dụ 9

2.1.1 Dịch và chạy ODE các ví dụ 10

2.2 Sử dụng 10

KHÁI NIỆM 11

3.1 Các khái niệm nền tảng 11

3.2 Vật rắn 11

3.2.1 Nhóm Islands và vật bị tắt 12

3.3 Tích phân hệ phương trình mô tả động lực học 12

3.4 Lực tác động lên vật 12

3.5 Khớp và ràng buộc 13

3.6 Nhóm khớp 13

3.7 Lỗi khớp và các tham số giảm lỗi 13

3.8 Ràng buộc mềm và lực rằng buộc 14

3.8.1 Lực rằng buộc 14

3.8.2 Cách dùng ERP và CFM 15

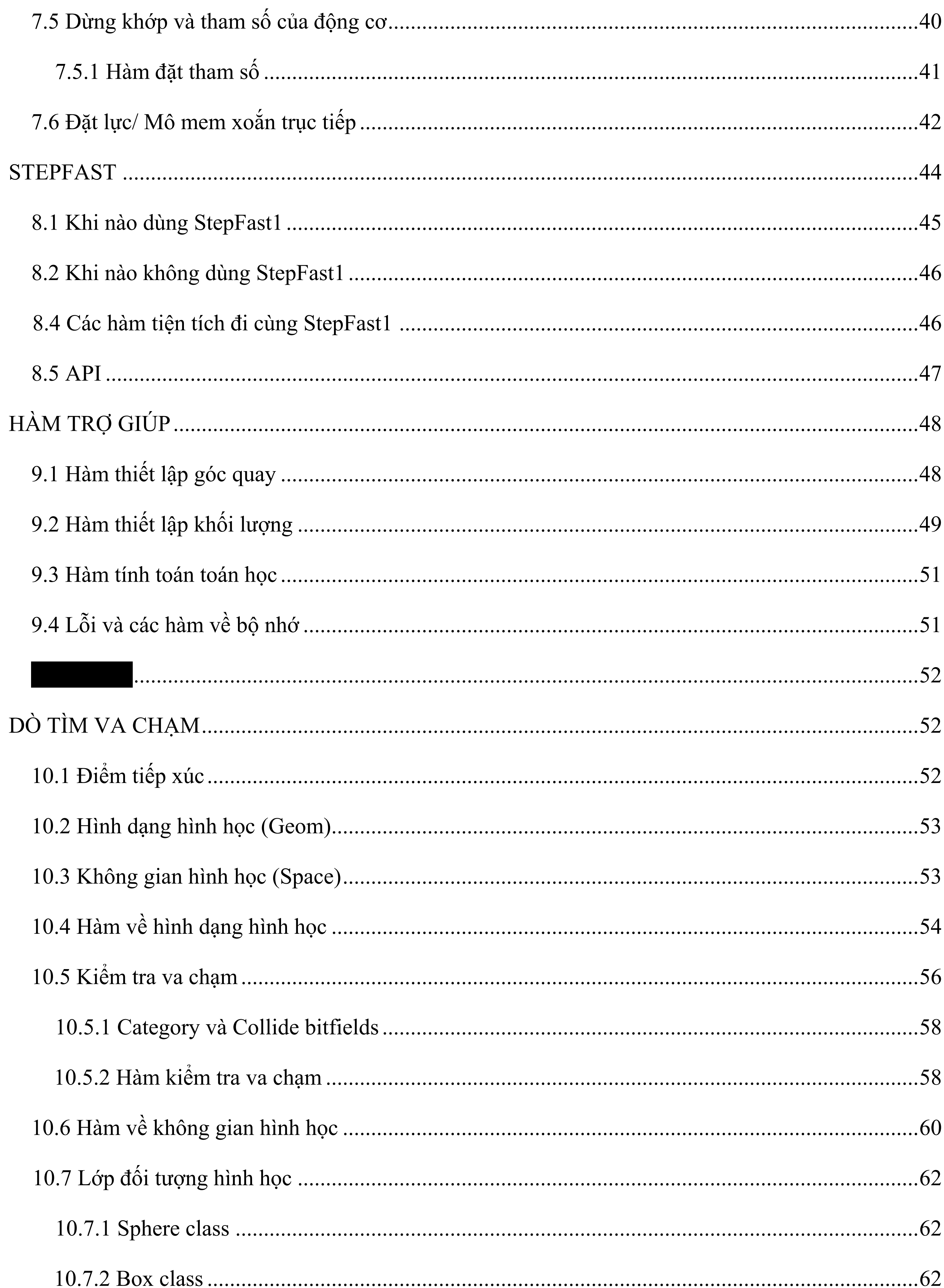
3.9 Kết quả va chạm 16

3.10 Quy trình khởi tạo hệ thống mô phỏng 16

3.11 Mô hình vật lý 17

3.11.1 Xấp xỉ ma sát 17

KIỂU DỮ LIỆU VÀ QUY TẮC 18
4.1 Kiểu dữ liệu cơ bản 18
4.2 Đối tượng và số hiệu định danh 18
4.3 Tham số chuyền 18
4.4 Ngôn ngữ C và C++ 19
4.5 Debuging 19
KHÔNG GIAN MÔ PHỎNG 20
ĐỘNG LỰC HỌC VẬT RẮN 22
6.1 Khởi tạo và hủy bỏ 22
6.2 Vị trí và hướng 22
6.3 Khối lượng và hệ lực 22
6.4 Hàm tiện ích 24
6.5 Hàm trạng thái 24
KHỚP ĐỘNG LỰC HỌC 26
7.1 Khởi tạo và hủy bỏ 26
7.2 Hàm trạng thái 27
7.3 Tham số của khớp 29
7.3.1 Khớp cầu 29
7.3.2 Khớp bản lề 29
7.3.3 Khớp trượt 30
7.3.4 Khớp Các – đăng 31
7.3.5 Khớp quay hai bậc tự do 32
7.3.6 Khớp cố định 33
7.3.7 Tiếp xúc 34
7.3.8 Khớp quay (AMotor) 37
7.4 Tham số chung 39



7.5 Dừng khớp và tham số của động cơ 40

7.5.1 Hàm đặt tham số 41

7.6 Đặt lực/ Mô mem xoắn trục tiếp 42

STEPFAST 44

8.1 Khi nào dùng StepFast1 45

8.2 Khi nào không dùng StepFast1 46

8.4 Các hàm tiện tích đi cùng StepFast1 46

8.5 API 47

HÀM TRỢ GIÚP 48

9.1 Hàm thiết lập góc quay 48

9.2 Hàm thiết lập khối lượng 49

9.3 Hàm tính toán toán học 51

9.4 Lỗi và các hàm về bộ nhớ 51

.... 52

DÒ TÌM VA CHẠM 52

10.1 Điểm tiếp xúc 52

10.2 Hình dạng hình học (Geom) 53

10.3 Không gian hình học (Space) 53

10.4 Hàm về hình dạng hình học 54

10.5 Kiểm tra va chạm 56

10.5.1 Category và Collide bitfields 58

10.5.2 Hàm kiểm tra va chạm 58

10.6 Hàm về không gian hình học 60

10.7 Lớp đối tượng hình học 62

10.7.1 Sphere class 62

10.7.2 Box class 62

10.7.3 Plane class 62
10.7.4 Capped cylinder class 63
10.7.5 Ray class 63
10.7.6 Triangle Mesh class 64
10.7.7 Geometry Transform class (Biến đổi hình học) 68
10.8 Đối tượng người dùng tự định nghĩa (User defined classes) 69
10.9 Đối tượng đa hợp (Composite objects) 71
10.10 Hàm tiện tích 72
10.11 Chú ý khi mô phỏng cơ hệ lớn 73
10.11.1 Khi cơ hệ lớn. 73
10.11.2 Dùng các thư viện khác nhau về va chạm 73
CÁC CHÚ Ý KHI XÂY DỰNG CƠ HỆ 75
11.1 Độ chính xác và độ ổn định của cơ hệ 75
11.2 Ứng xử cơ hệ phụ thuộc vào bước tính (step size) 75
11.3 Làm cho vật chuyển động nhanh hơn 75
11.4 Làm cho cơ hệ ổn định hơn 76
11.5 Tránh sự xuất hiện siêu liên kết 76
11.6 Các yếu tố khác 77
CÁC CÂU HỎI THƯỜNG GẶP 78
12.1 Làm thế nào nối một vật với giá? 78
12.2 Tại sao các vật bị nảy hoặc thâm nhập vào nhau khi va chạm? Kết quả nhận được là zero! 78
12.4 Cách thức tạo vật không di chuyển? 79
12.5 Vì sao lại phải đặt giá trị của ERP nhỏ hơn một? 79
12.6 Đặt trực tiếp vận tốc ban đầu cho vật, hay đặt lực hoặc mô men lực? 79
12.7 Tại sao, khi đặt vận tốc trực tiếp cho vật, thì tốc độ xử lý chậm hơn khi nối vật nối tới những thân thể khác? 80

12.8 Tôi có nên đặt đơn vị tỉ lệ xấp xỉ bằng 1 không? 80
12.9 Tôi mô phỏng xe ô tô, nhưng các bánh xe không đáp ứng chức năng! 80
12.10 Tôi làm như thế nào để có tương tác va chạm một chiều 81
12.12 Vật quay không ổn định! 81
12.13 Khi các vật quay đối khi bị kẹt giữa các geoms 82
12.13.1 Các vấn đề..83
12.13.2 Cách giải quyết 83

Chương 1

GIỚI THIỆU

Thư viện ODE là thư viện dùng để phát triển các sản phẩm phần mềm mô phỏng hệ nhiều vật, có mã nguồn mở. Ví dụ như mô phỏng chuyển động của xe trên bề mặt địa hình, các hệ chuyển động có khớp, và các vật chuuyển động trong không gian thực tại ảo (VR). Có tốc độ xử lý nhanh, có tính toán va chạm giữa các vật. ODE được phát triển bởi Russell Smith và được Trung tâm CNMP tiếp thu phát triển trong các ứng dụng mô phỏng của trung tâm.

1.1 Tính năng của phiên bản hiện thời

ODE dùng để tính toán mô phỏng động lực học của các hệ nhiều vật. Các hệ này có cấu trúc động học hình cây hay cấu trúc mạch khép kín nghĩa là: Các vật có thể được nối với nhau bằng các khớp lý tưởng có thể tạo thành các mạch động học khép kín. Ví dụ như Xe chuyển động trên mặt địa hình (Chỗ các bánh nối với khung dầm xe), Các vật thể chuyển động có chân (Chỗ các chân nối với thân), các liên kết giữa các chi tiết vũ khí trong quân sự.

ODE được thiết kế sao cho các nhà phát triển có thể dùng để mô phỏng hệ trong thời gian thực hoặc đủ nhanh để điều khiển trạng thái của hệ. Đặc biệt các vật chuyển động trong không gian thực tại ảo, kể cả khi người dùng thay đổi tùy ý các tham số đầu vào đặc trưng của hệ.

ODE sử dụng bộ tích phân toán học có độ ổn định cao, cho nên lỗi sinh ra khi mô phỏng sẽ không tăng khí điều khiển. Ý nghĩa vật lý của điều này là hệ thống mô phỏng sẽ không bị phá vỡ trong mọi lý ro (mà điều này rất dễ xẩy ra với các hệ mô phỏng nếu không cẩn thận).

ODE có khớp cứng. Nghĩa là không có sự tiếp xúc xuyên suốt khi va chạm giữa hai vật. Trong nhiều trường hợp nó được sử dụng như các vật trung gian thay thế, ví dụ như các lò xo ảo được dùng miêu tả tiếp xúc đó. Nó thật khó để làm đúng và giảm thiểu sai số xẩy ra khi mô phỏng.

ODE một xây dựng các hàm bắt va chạm. Tuy nhiên bạn có thể không quan tâm đến nó mà tự xây dựng hàm bẫy va chạm của riêng mình nếu bạn muốn. Hiện tại đã có các hàm bắt va chạm của một số các dạng hình học điển hình như hình hộp, hình trụ, mặt phẳng, và bề mặt tam giác- nhiều đối tượng khác sẽ được phát triển sau. Hệ thống các hàm bắt va chạm của ODE sẽ được cung cấp để nhận ra một cách nhanh chóng các mặt giao nhau của các đối tượng, thông qua những khái niệm về không gian mô phỏng.

Sau đây là các đối tượng được dùng trong thư viện:

- Vật rắn với khối lượng phân bố tùy ý
- Kiểu khớp nối: Khớp cầu, Khớp bản lề, Khớp trượt, Khớp Các – đăng, Khớp trượt-2, Khớp cố định, Tiếp xúc, Khớp trụ…
- Va chạm: Hình cầu, Hình trụ, Hình Hộp, Mặt tam giác, các tia thẳng
- Kiểu không gian va chạm: kiểu sân trong (gần như sân có bốn cạnh xung quanh), kiểu không gian hỗn độn, kiểu đơn giản.
- Phương pháp mô phỏng: Các phương trình chuyển động nhận được từ phương trình Lagrange nhân hệ vector dựa trên mô hình theo Trinkle/Stewart và Anitescu/Potra.
- Tích phân bậc nhất được sử dụng trong tính toán. Kết quả nhận được nhanh nhưng không đủ chính xác về định lượng. Các tích phân bậc cao sẽ được sử dụng trong thư viện này.

CÀI ĐẶT VÀ SỬ DỤNG

2.1 Cài đặt

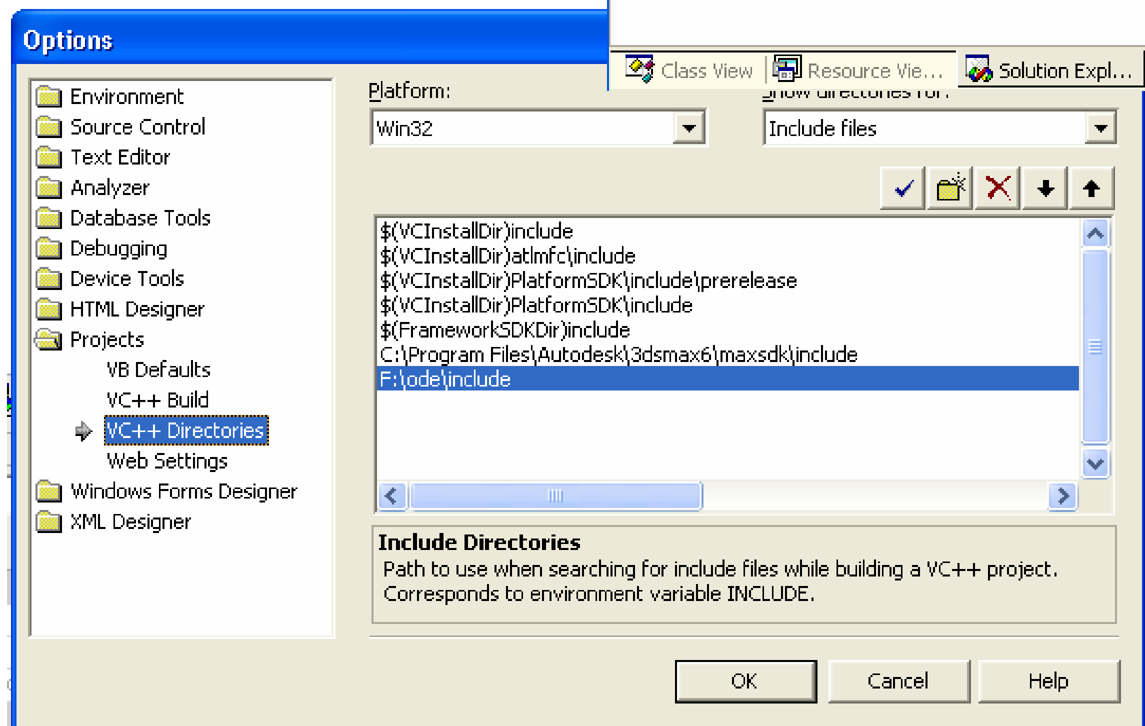
Bước 1: Giải nén gói thư viện ODE vào thư mục nào đó:

Ví dụ: F:\ODE

Bước 2: Nếu dùng VC để biên dịch thư viện thì cần thiết lập thông số trong bảng Option của menu tools …

Chạy file f:\ode\vc6\configure-single.bat hoặc copy file vc6_config\config-single.h vào thư mục f:\ode\\include\ode rồi đổi tên thành config.h.

Mở file f:\ode\vc6\ode.dsw để dịch thư viện.



2.1.1 Dịch và chạy ODE các ví dụ

Các ví dụ di cùng có trong thứ viện ta có tìm thấy trong thư mục vc6\Samples. Hãy chọn file MakeAllTests.dsw nếu muốn dích tất cả ví dụ.

Muốn chạy thử hãy copy thư mục f:\ode\drawstuff vào thư mục f:\ode\vc6\

2.2 Sử dụng

Cách thức tốt nhất để dùng thư viện này là xem mẫu các ví dụ đi cùng với thư viện. Nhưng có các chú ý sau:

- Khi lập trình thì chỉ cần đưa file ode.h vào theo lệnh:

  # Include <ode/ode.h>

- Với các cơ hệ lớn khi mô phỏng cần dùng lệnh dWorldStep() để khi chạy cần giải phóng bộ nhớ trách hiện tượng chàn bộ nhớ hoặc phá vỡ dữ liệu, chúng ta có thể dùng lệnh dWorldStepFast1() cũng cho kết quả tương tự.

Chương 3

KHÁI NIỆM

3.1 Các khái niệm nền tảng

3.2 Vật rắn

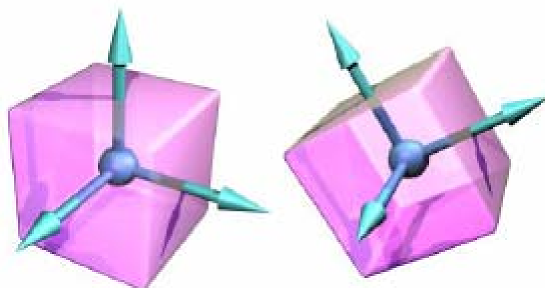
Vật rắn có nhiều thuộc tính từ vật điểm khi mô phỏng. Có nhiều thuộc tính sẽ thay đổi theo thời gian như sau:

- Ví trí (x,y,z) – điểm tham chiếu của vật hiện thời được xác định là trọng tâm của vật
- Vận tốc dài của vật điểm là vector (vx,vy,vz).
- Hướng của vật được miêu tả bởi ma trận quay 3x3
- Vận tốc góc là vector (wx,wy,wz) miêu tả hướng của vật thay đổi theo thời gian.

Các thuộc tính khác không thay đổi theo thời gian như:

- Khối lượng vật
- Trọng tâm của vật
- Ma trận vector quán tính của vật – là ma trận 3x3 miêu tả trọng lượng phân bố trên vật

Có thêm một khái niệm là các vật rắn có tọa độ x, y, z được gắn chặt vào vật và tọa độ này chuyển động, quay theo vật như **hình 3.1**



Hình 3.1 (Gốc tọa độ riêng của vật)

Gốc tạo độ này chính là điểm tham chiếu của vật, có rất nhiều giá trị có liên quan đến tọa độ này và một số khác có quan hệ với gố tọa độ tuyệt đối.

3.2.1 Nhóm Islands và vật bị tắt

Các vật rắn được nối kết với nhau bằng khớp, khái niệm “Island” của những vật rắn là nhóm các vật tương tác nhau không thể tách rời.

Với mỗi vật có thể được kích hoạt hoặc tắt. Khi bị tắt vật sẽ không được cập nhật vào không gian mô phỏng khi tính toán mục đích là để hủy bỏ các vật không còn vai trò trong các bước mô phỏng tiếp theo.

Nếu có nhiều vật được kích hoạt trong Island thì tất cả các vật sẽ vẫn được kích hoạt trong bước mô phỏng tiếp theo. Tương tự các vật bị tắt trong Island sẽ vẫn bị tắt. Nhưng nếu Island đã bị tắt mà chạm vào vật đang được kích hoạt thì Island này sẽ được kích hoạt theo, sinh ra khớp tiếp xúc giữa vật với island.

3.3 Tích phân hệ phương trình mô tả động lực học

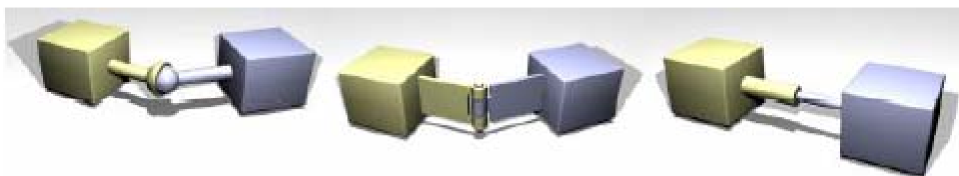
Quá trình mô phỏng hệ nhiều vật theo thời gian gọi là tích phân hệ phương trình mô tả động lực học hệ nhiều vật. Mỗi bước tích phân được tính từ thời gian hiện tại cộng thêm một khoảng thời gian phụ trội. Lúc này các thuộc tính của vật sẽ được tính toán theo các giá trị mới của thời gian. Sẽ có hai vấn đề lẩy sinh khi tích phân hệ phương trình mô tả động lực học bất kỳ.

- Sai số sẽ là bao nhiêu: Nghĩa là sự khác nhau giữa hệ mô phỏng với hệ thực sẽ như thế nào?.
- Độ ổn định của hệ mô phỏng: Liệu hệ mô phỏng có bị đổ vỡ bởi bất kì lý do nào không?

Hiện nay ODE đã có các bước tính rất ổn định, nhưng các bước tính toán chưa đủ nhỏ để giảm thiểu sai số. Việc dùng ODE vào việc mô phỏng kiểm nghiệm tính toán cho các chuyên ngành vẫn còn ở phía trước.

3.4 Lực tác động lên vật

Giữa các bước tính người dùng có thể gọi các hàm cập nhật các lực tác động lên vật các lực này được cộng vào lực tác động lên vật trong bước mô phỏng tiếp theo. Tổng lực này sẽ triệt tiêu sau mỗi bước tính.



Hình 3.2: Ba kiểu liên kết khác nhau

3.5 Khớp và ràng buộc

Trong thực tế có rất nhiều khớp có dạng bản lề. Chúng dùng để nối hai vật và được sử dụng trong ODE tạo nên sự quan hệ rằng buộc theo quy luật của hai đối tượng. Các thuộc tính đặc trưng thay đổi khi mô phỏng như vị trí, hướng sẽ quan hệ lẫn nhau. Sự liên hệ đó được gọi là rằng buộc có thể xem ở **hình 3.2** – là ba kiểu rằng buộc khác nhau.

Kiểu thứ nhất kiểu rằng buộc cầu (Khớp cầu) có ba bậc tự do quay quanh ba trục, kiểu thứ hai là khớp bản lề có một bậc tự do quay quanh một trục, kiểu thứ ba là khớp trượt một bậc tự do hai vật chuyển động tương đối với nhau theo một trục.

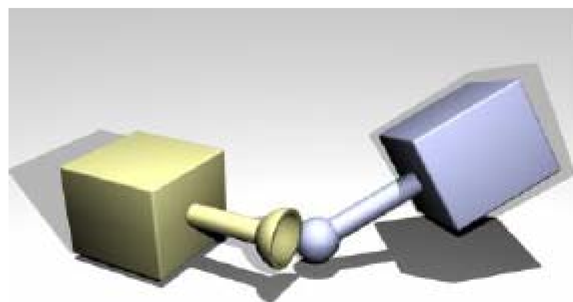
3.6 Nhóm khớp

Nhóm khớp là đối tượng đặc biệt chứa các khớp trong không gian mô phỏng. Mỗi khớp có thể gia nhập vào một nhóm khớp. Khi không cần dùng đến các khớp trong nhóm thì có thể hủy bỏ nhanh chóng bằng một hàm. Tuy nhiên, một khớp riêng lẻ không thể hủy bỏ được khi đối tượng chứa chưa rỗng. Nhóm khớp có hữu ích khí dùng các khớp tiếp xúc, khi thêm và hủy bỏ vào không gian mô phỏng cần rất nhiều bước để thực hiện.

3.7 Lỗi khớp và các tham số giảm lỗi

Khi dùng khớp nối hai vật, thì yêu cầu hai vật này phải có vị trí và hướng có quan hệ tươgn ứng với nhau. Tuy nhiên, có những vị trí mà hai vật chưa thực sự tạo được khớp nối sẽ gây ra lỗi. Có hai trường hợp sau:

i. Nếu thiết lập vị trí/ hướng của vật này không tương thích với vị trí/ hướng của vật còn lại



Hình 3.3 Ví dụ lỗi ở khớp cầu

ii. Trong khi mô phỏng. Lỗi có thể sinh ra cùng kết quả khi vật chuyển động tới vị trí yêu cầu

Xem **hình 3.3** là lỗi ở khớp cầu (khi mà tâm bi và tâm của ổ bi không trùng nhau)

Những kỹ thuật giảm lỗi: Ở mỗi bước mô phỏng mỗi khớp sẽ có những lực đặc biết tác động lên làm cho các vật được căn lại đúng vị trí. Những lực này được điều khiển bởi các tham số giảm lỗi (ERP) các tham số này có giá trị từ 0 tới 1.

ERP sẽ phải có giá trị nào đó để điều chỉnh lỗi khớp trên mỗi bước mô phỏng. Nếu ERP = 0 không cần có lực nào được cập nhật vào khớp quá trình mô phỏng vẫn được tiếp tục. Nếu ERP = 1 thì quá trình mô phỏng ở bước tiếp theo sẽ cập nhật lực để hiệu chỉnh lỗi trên các khớp. Tuy nhiên cũng khuyến cáo không nên dùng ERP = 1 bởi ở giá trị này lỗi ở các khớp sẽ không được chỉnh sửa hoàn toàn do các bước xấp xỉ để tính toán bên trong thuật toán. Các giá trị từ 0.1 đến 0.8 được klhuyến cáo nên dùng (Ở đây giá trị mạc định là 0.2).

Giá trị toàn cục ERP có thể được thiết lập để đặt cho tất cả các khớp trong không gian mô phỏng. Tuy nhiên một vài khớp đặc biệt có thể đặt riêng giá trị ERP.

3.8 Ràng buộc mềm và lực rằng buộc

Trong bài toán thực tế đa số các khớp là liên kết cứng nghĩa là các điều kiện tạo liên kết khôgn bao giờ được vi phạm. Ví dụ như Khớp cầu bi cầu luôn ở trong ổ cầu, hai phần của khớp bản lề luôn nằm trên cùng một trục. Trong khi thiết lập có thể bị vi phạm mang tính vô ý tạo lỗi cho hệ thống. Nhưng tham ERP sẽ thiết lập và khắc phục điều này.

Không phải tất cả các rằng buộc đều là rằng buộc cứng. Có một vài kiểu rằng buộc mềm được thiết kế để dử dụng. Ví dụ sự rằng buộc tiếp xúc là giằng buộc cứng được tạo lập để ngăn ngừa va chạm giữa các vật, như va chạm giữa các mặt có vật liệu bằng thép. Nhưng cũng có thể được thiết lập là rằng buộc mềm cho mô phỏng các vật được làm bằng các vật liệu mềm hơn, do đó miêu tả được quá trình va chạm tự nhiên giữa hai vật.

Có hai tham số điều khiển sự khác biệt giữa rằng buộc cứng và rằng buộc mềm. Tham số thứ nhất là ERP đã được đề cập. Tham số thứ hai là hỗn hợp lực rằng buộc (CFM) sẽ được đề cập ở dưới.

3.8.1 Lực rằng buộc

Theo phương trình truyền thống rằng buộc cho mọi khớp có dạng:

$$J^* v = c \tag{3.1}$$

Trong đó: *v* là vector vận tốc của vật, *J* ma trận Jacobian là ma trận bận tự do của khớp và c là vector bàn tay phải. Ở bước sau vector lambda được tính toán (có kích thước bằng *c)* sau đó lực được cập nhật tác động vào vật để giữ các rằng buộc theo công thức :

$$force = J^T * \lambda \qquad (3.2)$$

ODE sẽ công thêm một rằng buộc mới có phương trình là :

$$J * v = c + CFM * \lambda \qquad (3.3)$$

Trong đó : CFM là ma trận đường chéo. CFM chọn kết quả lực rằng buộc với rằng buộc sinh ra nó. Một giá trị khác không và dương của CFM cho phép phương trình rằng buộc gốc được đề cập đến bởi một lượng tương ứng cho thời gian CFM khôi phục lực λ một cách bắt buộc. Biến đổi phương trình ta được

$$(JM^{-1}J^T + CFM/h)\lambda = c/h \qquad (3.4)$$

Như vậy CFM thêm vào đường chéo của ma trận nguyên gốc hệ thống. Dùng giá trị dương của CFM đã cải thiện được độ chính xác.

3.8.2 Cách dùng ERP và CFM

ERP và CFM có thể được thiết lập độc lập đối với mỗi khớp. Chúng có thể được thiết lập với cả rằng buộc tiếp xúc, trong liên kêt có giới hạn và ở một vài vị trí khác.

Nếu CFM được đặc giá trị là không, rằng buộc chở thành rằng buộc cứng. Nếu CFM là giá trị dương thì rằng buộc sẽ có độ mềm như rằng buộc bằng lực giữa hai đối tượng tiếp xúc.Độ mềm dẻo của rằng buộc sẽ tăng khi ta tăng hệ số CFM. Chú ý là nếu thieté lạp giá trị âm cho CFM sẽ có hiệu ứng xấu xẩy ra tạo cho hệ thống mất ổn định. Không nên dùng giá trị này.

Các giá trị của ERP và CFM ta có thể đạt được nhiều hiệu quả khác nhau. Ví dụ như tạo rằng buộc đàn hồi, nơi hai vật được nối với nhau bằng lò xo. Thực ra ERP và CFM có thể được chọn lựa như các hệ số của lò xo và giảm chấn. Nếu bạn có hệ số cứng của lò xo là k_p và hệ số giảm chấn k_d thì phương trình tính toán tương ứng trong thư viện sẽ là :

$$ERP = hk_p/(hk_p + k_d) \qquad (3.5)$$

$$CFM = 1/(hk_p + k_d) \qquad (3.6)$$

Trong đó : h là bước tính

Khi tăng CFM, đặc biệt giá trị toàn cục CFM sẽ giảm sai số tính toán trong mô phỏng. Nếu là hệ thống đơn thì làm tăng thêm độ ổn định. Nếu hệ thống không hoạt động ta hãy thử tăng giá trị của CFM toàn cục.

3.9 Kết quả va chạm

Sự va chạm giữa các vật hoặc giữa các vật với môi trường nhận được như sau :

1. Trước mỗi bước mô phỏng, người dùng gọi hàm kiểm tra va chạm để kiểm tra cái gì va chạm với cái gì. Hàm này trả về danh sách các điểm tiếp xúc. Mỗi điểm tiếp xúc được chỉ rõ vị trí trong không gian, và vector chỉ phương trên mặt.
2. Rằng buộc tiếp xúc tạo ra các điểm tiếp xúc tương ứng và nhận thêm thông tin về xự tiếp xúc giữa các bề mặt. Ví dụ Ma sát giữa các mặt thì độ lẩy và độ mềm là như thế nào và các thuộc tính khác.
3. Rằng buộc tiếp xúc được đặt vào trong một nhóm điều này cho phép khởi tạo thêm và hủy bỏ chúng ra khỏi hệ thống một cách nhanh chóng. Tốc độ mô phỏng giảm đi khi các điểm tiếp xúc trên khớp tăng lên. Vì vậy cần có cách giới hạn số điểm tiếp xúc trên mỗi khớp.
4. Bước mô phỏng cần được quan tâm
5. Tất cả khớp tiếp xúc cần được hủy bỏ khỏi hệ thống

Chú ý : Không phải tất cả các hàm tìm va chạm trong thư viện đều cần dùng. Ta có thể dùng các hàm va chạm tự viết khác miễn sao bảo đảm được những thông tin về khớp tiếp xúc.

3.10 Quy trình khởi tạo hệ thống mô phỏng

Quy trình khởi tạo hệ thống mô phỏng theo các bước như sau :

1. Tạo không gian đọng lực học
2. Tạo các vật thể trong không gian mô phỏng
3. Thiết lập trạng thái của vật
4. Tạo các khớp động lực học
5. Gắn các khớp vào các vật thể
6. Xác định các điều kiện đầu cho mỗi khớp
7. Tạo hàm kiểm tra va chạm và hình dạng hình học của đối tượng để kiểm tra va chạm, điều này rất quan trọng.
8. Tạo nhóm khớp để chứa các khớp tiếp xúc
9. Vòng lặp

a. Thêm các lực tác động vào vật

b. Điều chỉnh tham số của khớp

c. Gọi hàm kiểm tra va chạm

d. Tạo các khớp tiếp xúc cho tất cả các điểm tiếp xúc va chạm rồi đưa chúng vào nhóm khớp tiép xúc

e. Thiết lập bước mô phỏng

f. Hủy bỏ tất cả các khớp tipé xúc ra khỏi hệ thống

10. Hủy bỏ hệ thống mô phỏng và khôgn gian kiểm tra va chạm.

3.11 Mô hình vật lý

Một số phương pháp và tính toán được bàn luận ở đây.

3.11.1 Xấp xỉ ma sát

Mô hình ma sát culong là mô hình đơn giản nhất. Nhưng rất hiệu quả để tính toán ma sát tại những điểm tiếp xúc. Nó là mối quan hệ giữa lực tiếp tuyến và phát tuyến ở điểm tiếp xúc (Xem ở phần mô tả khớp tiếp xúc có miêu tả lực này) theo công thức:

$$|f_T| <= \mu * |f_N| \qquad (3.7)$$

Trong đó f_T và f_N là vector lực pháp tuyến và tiếp tuyến điểm tiếp xúc, μ là hệ số ma sát (Có giá trị trong khoảng 1). Phương trình này định nghĩa một hình nón ma sát, tưởng tượng có hình nón mà f_N là trục và các điểm tiếp xúc là những đỉnh. Nếu tổng vector lực ma sát nằm trên hình nón

Chương 4

KIỂU DỮ LIỆU VÀ QUY TẮC

4.1 Kiểu dữ liệu cơ bản

Thư viện ODE có thể biên dịch ở hai chế độ khác nhau float hoặc double. Khi dùng float thì cần ít bộ nhớ và tốc độ tính toán nhanh hơn. Nhưng sai số mô phỏng sẽ cao và kết quả nhiều khi có vấn đề.

Kieủe dữ liệu float được định nghĩa là dReal và những kiểu thường sử dụng khác là dVector3, dVector4, dMatrix3, dMatrix4, dQuanternion

4.2 Đối tượng và số hiệu định danh

Có nhiều kiểu đối tượng có thể được dùng

- dWorld – không gian mô phỏng động lực học
- dSpace – Không gian hình học (Có tính đến va chạm)
- dBody – Vật rắn
- dGeom – Hình dạng hình học của vật rắn (Để kiểm tra va chạm)
- dJoint – Khớp động lực học
- dJointGroup – Nhóm khớp

Khi khởi tạo các hàm khởi tạo trả về các chỉ số định danh IDs. Kiểu định danh đối tượng như dWorldID, dBodyID…

4.3 Tham số chuyền

Tất cả các vector3 đều cung cấp các hàm thiết lập "set" và có thể thiết lập riêng lẻ các thuộc tính x, y, z. Tất cả các vector3 đều cung cấp các hàm lấy giá trị thuộc tính "get" hàm này trả về mảng dữ liệu dReal.

Các dạng vector có kích thước lớn hơn đều cung cấp các hàm tương tự nhưng với mảng dữ liệu lớn hơn.

Tất cả các hệ tọa khi khởi tạo đều là tọa độ tuyệt đối loại chừ khi được chỉ rõ.

4.4 Ngôn ngữ C và C++

Thư viện được viết bằng ngôn ngữ C++, nhưng các hàm toàn cụ được viết bằng ngôn ngữ C thường. Tại sao lại như vậy?

- Khi dùng C ngôn ngữ đơn giản hơn và một số chức năng thư viện cần trợ giúp không được C++ hỗ trợ
- C++ hạn chế các trình biên dịch
- Các nhà phát triển không dùng C++ cũng sử dụng được thư viện.

4.5 Debuging

Thư viện có thể được biên dịch ở cả hai chế độ “debug” và “release”. Ở chế độ “debuging” thư viện chạy chậm hơn nhưng có chế độ kiểm tra các hàm một cách chắc chắn. Chế độ “release” thư viện chạy nhanh hơn nhưng không có chế độ kiểm tra các hàm.

Chương 5

KHÔNG GIAN MÔ PHỎNG

Không gian mô phỏng động lực học là không gian chứa các vạt rắn và các khớp. Các đối tượng ở các không gian khác nhau thì không thể tương tác với nhau. Ví dụ hai vật trong hai không gian khác nhau thì không thể va chạm với nhau.

Tất cả đối tượng trong không gian tồn tại cùng một thời điểm. Như vậy một lý do sử dụng các không gian mô phỏng riêng lẻ sẽ mô phỏng ở các tốc độ khác nhau.

Phần lớn khi xây dựng các ứng dụng thường có một không gian mô phỏng.

```
dWorldID dWorldCreate();
```

Tạo mới không gian mô phỏng rỗng và trả về ID

```
void dWorldDestroy (dWorldID);
```

Hủy bỏ không gian mô phỏng và tất cả thuộc nó. Bao gồm tất cả các vật, tất cả các khớp và loại trừ nhóm khớp. Các nhóm này sẽ không được kích hoạt và loại bằng hàm `dJointGroupEmpty`.

```
void dWorldSetGravity (dWorldID, dReal x, dReal y, dReal z);
void dWorldGetGravity (dWorldID, dVector3 gravity);
```

là hai hàm thiết lập và lấy vector trọng trường. Có đơn vị là m/s/s, Với trái đất vector trọng trường là (0,0,-9.81), Nếu +z chiều hướng lên. Giá trị mạc định là (0,0,0).

```
void dWorldSetERP (dWorldID, dReal erp);
dReal dWorldGetERP (dWorldID);
```

là hai hàm thiết lập và lấy giá trị toàn cục ERP, Tham số điều khiển khả năng sửa lỗi của hệ thống sau mỗi bước mô phỏng. Thông thường giá trị này nằm trong khoảng 0.1–0.8. Giá trị mạc định là 0.2.

```
void dWorldSetCFM (dWorldID, dReal cfm);
dReal dWorldGetCFM (dWorldID);
```

là hai hàm thiết lập và lấy giá trị toàn cục CFM. Thông thường giá trị này nằm trong khoảng $10^{-9} - 1$. Giá trị mạc định là 10^{-5} nếu dùng ở chế độ float, hoặc 10^{-10} nếu dùng ở chế độ double.

```
void  dWorldSetAutoDisableFlag (dWorldID, int do_auto_disable);
int   dWorldGetAutoDisableFlag (dWorldID);
void  dWorldSetAutoDisableLinearThreshold (dWorldID, dReal linear_threshold);
dReal dWorldGetAutoDisableLinearThreshold (dWorldID);
```

```
void  dWorldSetAutoDisableAngularThreshold (dWorldID, dReal angular_threshold);
dReal dWorldGetAutoDisableAngularThreshold (dWorldID);
void  dWorldSetAutoDisableSteps (dWorldID, int steps);
int   dWorldGetAutoDisableSteps (dWorldID);
void  dWorldSetAutoDisableTime (dWorldID, dReal time);
dReal dWorldGetAutoDisableTime (dWorldID);
```

Hàm thiết lập và lấy giá trị mạc định của các biến auto-disable khi khởi tạo vật động lực học mới, sau đây là các giá trị mạc định của các tham số điều khiển:

- AutoDisableFlag = disabled
- AutoDisableLinearThreshold = 0.01
- AutoDisableAngularThreshold = 0.01
- AutoDisableSteps = 10
- AutoDisableTime = 0

`void dWorldStep (dWorldID, dReal stepsize);` Bước tính trong không gian mô phỏng

```
void dWorldImpulseToForce (dWorldID, dReal stepsize,dReal ix, dReal iy, dReal iz,
dVector3 force);
```

Nếu bạn dùng một xung tuyến tính thẳng hay xung góc vào vật, thay vì một lực hoặc mô mem, thì bạn có thể dùng hàm này để chuyển mong muốn thành lực/mô men trước khi gọi hàm dBodyAdd...Hàm này cho ra vector xung (ix,iy,iz) và đặt vector này thành vector lực. Hiện tại giải thuật được dùng là lấy tỉ lệ xung bằng 1/stepsize, trong đó stepsize là độ rộng của bước tính tiếp theo. Hàm này còn dùng tham số dWorldID bởi vì trong tương lai các lực được đưa vào còn phụ thuộc vào các tham số của không gian mô phỏng thực tế.

```
void dCloseODE();
```

Hàm này không thể giải phóng bộ nhớ. Bạn có thể dùng hàm `dWorldDestroy()` trược khi thoát khỏi chương trình ứng.

Chương 6

ĐỘNG LỰC HỌC VẬT RẮN

6.1 Khởi tạo và hủy bỏ

```
dBodyID dBodyCreate (dWorldID);
```

Khởi tạo vật rắn với các gia strị mạc định về khối lượng và vị trí đặt tại (0,0,0) trả lại ID

```
void dBodyDestroy (dBodyID);
```

Hủy bỏ vật. Tất cả các khớp được nối với vật này sẽ được đặt lại vào trong “limbo” (không nối và không gây ảnh hưởng đến mô phỏng nhưng vẫn không bị hủy bỏ khỏi không gian mô phỏng)

6.2 Vị trí và hướng

```
void dBodySetPosition   (dBodyID, dReal x, dReal y, dReal z);
void dBodySetRotation   (dBodyID, const dMatrix3 R);
void dBodySetQuaternion (dBodyID, const dQuaternion q);
void dBodySetLinearVel  (dBodyID, dReal x, dReal y, dReal z);
void dBodySetAngularVel (dBodyID, dReal x, dReal y, dReal z);
const dReal * dBodyGetPosition   (dBodyID);
const dReal * dBodyGetRotation   (dBodyID);
const dReal * dBodyGetQuaternion (dBodyID);
const dReal * dBodyGetLinearVel  (dBodyID);
const dReal * dBodyGetAngularVel (dBodyID);
```

Những hàm này là hàm thiết lập hoặc lấy giá trị về vị trí, góc quay, vận tốc thẳng, vận tốc góc của vật. Sau khi thiết lập nhóm vật rắn, kết quả là mô phỏng không hiểu rõ. Nếu các cấu hình thiết lập mâu thuẫn với khớp/rằng buộc đã có. Khi lấy giá trị, giá trị nhận được là một con trỏ có cấu trúc, như vậy những vector có tính hợp lệ cho đến khi bất kỹ sự thay đổi nào làm cho thay đổi kết cấu của vật rắn.

Ở đây dBodyGetRotation cho trả về ma trận 4x3.

6.3 Khối lượng và hệ lực

```
void dBodySetMass (dBodyID, const dMass *mass);
void dBodyGetMass (dBodyID, dMass *mass);
```

Những hàm này là hàm thiết lập hoặc lấy giá trị trọng lượng của vật (xem các hàm về trọng lượng).

```
void dBodyAddForce              (dBodyID, dReal fx, dReal fy, dReal fz);
void dBodyAddTorque             (dBodyID, dReal fx, dReal fy, dReal fz);
void dBodyAddRelForce           (dBodyID, dReal fx, dReal fy, dReal fz);
void dBodyAddRelTorque          (dBodyID, dReal fx, dReal fy, dReal fz);
void dBodyAddForceAtPos         (dBodyID, dReal fx, dReal fy, dReal fz,
                                          dReal px, dReal py, dReal pz);
void dBodyAddForceAtRelPos      (dBodyID, dReal fx, dReal fy, dReal fz,
                                          dReal px, dReal py, dReal pz);
void dBodyAddRelForceAtPos      (dBodyID, dReal fx, dReal fy, dReal fz,
                                          dReal px, dReal py, dReal pz);
void dBodyAddRelForceAtRelPos (dBodyID, dReal fx, dReal fy, dReal fz,
                                          dReal px, dReal py, dReal pz);
```

void dBodyAddRelForceAtRelPos (dBodyID, dReal fx, dReal fy, dReal fz, dReal px, dReal py, dReal pz);

Đặt thêm lực tác động vào vất (Theo tọa độ tuyệt đối hoặ tương đối) các lực được tích lũy tác động lên vật, và những lực này sẽ tự triệt tiêu sau mỗi bước tính.

Các hàm...RelForce và ...RelTorque đặt lực tác động lên vật theo hệ tọa độ cục bộ của vật.

Các hàm ...ForceAtPos và ...ForceAtRelPos đặt lực tác động vào vật tại một điểm (theo tọa độ tuyệt đối hoặc tương đối) đây là điểm đặc biệt để đặt lực. Còn tất cả các lực khác đều có điểm đặt tại trọng tâm của vật.

```
const dReal * dBodyGetForce (dBodyID);
const dReal * dBodyGetTorque (dBodyID);
```

Đây là hai hàm trả về lực và mô men tích tũy hiện thời trả về con trỏ kiểu dReal trỏ vào một mảng có độ rộng là 3. Những giá trị được trả lại là những con trỏ có cấu trúc giữ liệu bên trong. Như vậy giá trị hợp lệ của vector chỉ có giá trị đến khi bất kỳ sự thay đổi nào làm cho hệ thống thay đổi.

```
void dBodySetForce (dBodyID b, dReal x, dReal y, dReal z);
void dBodySetTorque (dBodyID b, dReal x, dReal y, dReal z);
```

Đặt lực/mômen tích lũy tác động lên vật. Các lực này sẽ triệt tiêu trong khi các vật không được kích hoạt, chúng chỉ có giá trị khi vật được kích hoạt trở lại. Trong nhiều trường hợp hàm đặt lực được gọi trong khi các vật không được kích hoạt.

6.4 Hàm tiện ích

```
void dBodyGetRelPointPos (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
void dBodyGetRelPointVel (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
void dBodyGetPointVel (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
```

Các hàm tiện ích lấy một điểm thuộc vật (px, py, pz) trả về tọa độ tuyệt đối hoặc vận tốc của điểm đó hàm dBodyGetRelPointXXX trả điểm về tọa độ cụ bộ của vật dBodyGetPointVel trả điểm về tọa độ toàn cầu.

```
void dBodyGetPosRelPoint (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
```

Hàm này ngược với hàm `dBodyGetRelPointPos()` lấy điểm trong hệ tọa độ toàn cầu (x,y,z) trả về tọa độ cục bộ trên vật.

```
void dBodyVectorToWorld (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
void dBodyVectorFromWorld (dBodyID, dReal px, dReal py, dReal pz,dVector3 result);
```

Lấy vector trên vật (Trên hệ tọa độ toàn cầu) có tọa độ (x,y,z), đưa vào tọa độ trên vật (toạn độ toàn cầu) .

6.5 Hàm trạng thái

```
void dBodySetData (dBodyID, void *data);
void *dBodyGetData (dBodyID);
```

Đây là hai hàm đặt/và lấy dữ liệu của vật có kiểu con trỏ void* .

```
void dBodyEnable (dBodyID);
void dBodyDisable (dBodyID);
int dBodyIsEnabled (dBodyID);
```

Bật tắt vật thể. Nếu vataạ không được kích hoạt các thuộc tính của vật sẽ không được cập nhật trong các bước tính toán. Tuy nhiên nếu vật không được kích hoạt ra nhập vào “island” mà có chứa các vật đang được kích hoạt thì vật này sẽ được tái kích hoạt trong bước tính toán tiếp theo.

Hàm dBodyIsEnable(dBody ID) trả về giá trị 1 nếu vật được kích hoạt và 0 nếu không được kích hoạt. Khi khởi tạo vật mới vật này ở chế độ kích hoạt.

```
void dBodySetFiniteRotationMode (dBodyID, int mode);
```

Hàm này điều khiển cách thức cập nhật hướng của vật ở các bước mô phỏng giá trị mode có thể là:

0: Chế độ dành cho phép tính toán nhanh "infinitesimal", nhưng thỉnh thoảng gây ra sự không chính xác trong tính toán ở các vật có tốc độ quay cao. Đặc biệt là ở những vật được nối với các vật khác. Chế độ này là mạc định cho các đối tượng được khởi tạo mới.

1: Chế độ hạn chế về tốc độ tính toán "finite", nhưng chính xác đối với vật có tốc độ quay nhanh. Tuy nhiên khi tốc độ quay nhanh thỉnh thoảng kết quả vẫn nhận được lỗi không mong muốn.

```
int dBodyGetFiniteRotationMode (dBodyID);
```

Trả về giá trị hiện thời giới hạn quay của vật (0 or 1).

```
void dBodySetFiniteRotationAxis (dBodyID, dReal x, dReal y, dReal z);
```

Thiết lập trục giới hạn góc quay. Trục này chỉ có giá trị khi dùng chế độ "finite" xem hàm dBodySetFiniteRotationMode(). Nếu trục này là (0,0,0), chế độ "finite" sẽ được thực hiện đầy đủ đối với vật thể. Nếu trục này khác (0,0,0) thì vật quay thực hiện quay ở chế độ "finite" quanh trục mà ở chế độ "infinitesamal" trục quay trực giao với nó.

Các hàm này có hữu ích để giảm nguồn lỗi gây ra bởi các vật quay tròn quanh trục. Ví dụ nếu bánh xe quay ở tốc độ cao bạn có thể gọi hàm này cho khớp bản lề ở bánh xe để cải thiện kết quả của nó.

```
void dBodyGetFiniteRotationAxis (dBodyID, dVector3 result);
```

Hàm hoàn trả trục quay "finite" của vật thể.

```
int dBodyGetNumJoints (dBodyID b);
```

Hàm hoàn trả số lượng khớp nối với vật.

```
dJointID dBodyGetJoint (dBodyID, int index);
```

Hàm hoàn trả khớp nối với vật thể có chỉ số là index nằm trong khoảng 0 – n-1 trong đó n = `dBodyGetNumJoints().`

```
void dBodySetGravityMode (dBodyID b, int mode);
int dBodyGetGravityMode (dBodyID b);
```

Hàm thiết lập và lấy giá trị của ảnh hưởng trọng trường đến vật. Nếu mode=0 sẽ không có sự ảnh hưởng. Nếu mode=1 sẽ chịu sự tác động. Khi khởi tạo vật mạc định là vật luôn chịu tác động của trọng trường.

Chương 7

KHỚP ĐỘNG LỰC HỌC

7.1 Khởi tạo và hủy bỏ

```
dJointID dJointCreateBall (dWorldID, dJointGroupID);
dJointID dJointCreateHinge (dWorldID, dJointGroupID);
dJointID dJointCreateSlider (dWorldID, dJointGroupID);
dJointID dJointCreateContact (dWorldID, dJointGroupID,
                              const dContact *);
dJointID dJointCreateUniversal (dWorldID, dJointGroupID);
dJointID dJointCreateHinge2 (dWorldID, dJointGroupID);
dJointID dJointCreateFixed (dWorldID, dJointGroupID);
dJointID dJointCreateAMotor (dWorldID, dJointGroupID);
```

Các hàm này là khởi tạo một kiểu khớp. Khi khởi tạo các khớp mạc định được gắn với vật thể "limbo" (vật này không gây ảnh hưởng đến không gian mô phỏng) bởi vì nó không nối với bất kỳ vật nào. Các khớp sẽ được đặt vào nhóm khớp mạc định có ID là 0 đây là mã nhóm dạnh nghĩa không tồn tại thực. Riêng kiểu khớp tiếp xúc "contact" được khởi tạo với giái trị mạc định theo cấu trúc dContact

```
void dJointDestroy (dJointID);
```

Hàm laọi bỏ khớp sẽ có tác dụng bỏ liên kết giữa chúng với vật thể và hủy bỏ chúng ra khỏi không gian mô phỏng. Tuy nhiên nếu khớp nằm trong một nhóm thì lệnh này không có hiệu quả, chỉ đến khi nhóm rỗng hoặc đã bị hủy bỏ.

```
dJointGroupID dJointGroupCreate (int max_size);
```

Khởi tọa một nhóm khớp. Tham số max_size hiện nay chưa sử dụng có giá trị là 0. Nó đang được cân nhắc trong các phiên bản tiếp theo.

```
void dJointGroupDestroy (dJointGroupID);
```

Hủy bỏ nhóm khớp. Tất cả các khớp thuộc nó cũng bị hủy bỏ theo.

```
void dJointGroupEmpty (dJointGroupID);
```

Làm rỗng nhóm khớp. Tất cả các khớp thuộc nó sẽ bị hủy bỏ. Nhưng khớp vẫn tồn tại khôgn bị hủy bỏ.

7.2 Hàm trạng thái

```
void dJointAttach (dJointID, dBodyID body1, dBodyID body2);
```

Nối khớp với các vật. Nếu khớp đã được nói với một vật nào đấy, thì trước hết nó sẽ bỏ nối kết với vật cũ rồi nối với vật mới. Nếu body1 hoặc body2 là 0 thì các vật này tham chiếu đến giá. Nếu cả hai là 0 thì nghĩa là khớp được đặt liên kết với vật "limbo" không tham gia vào quá trình mô phỏng.

```
void dJointSetData (dJointID, void *data);
```

```
void *dJointGetData (dJointID);
```

Hàm gán và lấy giá trị của khớp

```
int dJointGetType (dJointID);
```

hàm hoàn trả lại kiểu khớp. là giá trị tương ứng ở bảng sau:

Kiểu khớp	Tên khớp thực tế
dJointTypeBall	Kiểu khớp cầu ba bậc tự do
dJointTypeHinge	Kiểu khớp bản lề một bậc tự do
dJointTypeSlider	Kiểu khớp trượt một bậc tự do
dJointTypeContact	Kiểu khớp tiếp xúc (Mặt, đường, điểm)
dJointTypeUniversal	Kiểu khớp Các đăng
dJointTypeHinge2	Kiểu khớp trượt hai bậc tự do
dJointTypeFixed	Kiểu khớp cố định 0 tự do
dJointTypeAMotor	Khiểu khớp trụ

```
dBodyID dJointGetBody (dJointID, int index);
```

Hoàn trả vật nối với khớp. Tham số index nhận giá trị 0 hoặc 1. Nghĩa là lấy vật thứ nhất hay vật thứ hai. Nếu hàm hoàn trả về 0 nghĩa là khớp đang được nối với giá. Nếu cả hai giá trị của hàm trả về nều là 0 thì khớp được nôi svới "limbo" không ảnh hưởng đến không gian mô phỏng.

```
void dJointSetFeedback (dJointID, dJointFeedback *);
```

```
dJointFeedback *dJointGetFeedback (dJointID);
```

Ở mỗi thời điểm mô phỏng. Lực do khớp sinh ra được công thêm vào thành phàn lực tác động trược tiếp vào vật và ta không biết được lực này được phân bố như thế nào.

Nếu thông tin này cần quan tâm thì người dùng có thể cung cấp một kiểu dữ liệu dJointFeedback để truyền vào hàm JointSetFeedback(). Kiểu dữ liệu này được định nghĩa như sau:

```
typedef struct dJointFeedback {
      dVector3 f1; // L•c kh•p tác ••ng vào  body 1
```

```
    dVector3 t1; // Mô mem tác ••ng vào body 1
    dVector3 f2; // L•c kh•p tác ••ng vào  body 2
    dVector3 t2; // Mô mem tác ••ng vào body 2
} dJointFeedback;
```

Trong mỗi bước thời gian bất kỳ những cấu trúc phản hồi nối tác động vào vật từ khớp nhận được thông tin chung về lực và mô mem hàm dJointGetFeedback() trả về dữ liệu có cấu trúc cấu trúc dJointFeedback hoặc 0 nếu không được sử dụng (đây là giá trị mạc định). Hàm dJointSetFeedback() nếu thiết lập thì sẽ bỏ qua các lực khớp động.

Chú ý: Có một vài hướng dẫn làm tính đổi hỏi người dùng phải tuân thủ cấu trúc này. Tại sao không lưu dữ liệu dJointFeedback ở dạng tĩnh cho mỗi khớp? là vì không phải người dùng nào cũng quan tâm đến điều này. Ngay cả khi dùng cũng không phải tất cả các khới cần dùng. Thật là tốn bộ nhớ nếu làm điều này nhất là cấu trúc này có thể sẽ được bỏ sung thêm nhiều thành phần trong tương lai.

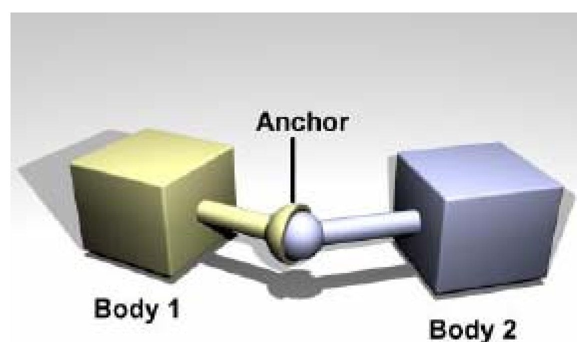
Tại sao trong thuộc tính của khớp không có kiểu dữ liệu này? Lý do là các khớp tiếp xúc (khi khởi tạo và hủy bỏ ở mỗi thời điểm) cần rất nhiều thời gian để đưa dữ liệu vào bộ nhớ nhất là khi dJointFeedback được yêu cầu. Ở đây cho phép người dùng tự bổ sung có vẻ là tốt hơn, đơn gian rlà chỉ cần cấp phát một mảng cố định.

```
int dAreConnected (dBodyID, dBodyID);
```

Đây là hàm tiện ích cho giá trị 1 nếu hai vật được nối với nhau qua một khớp, còn không trả về 0.

```
int dAreConnectedExcluding (dBodyID, dBodyID, int joint_type);
```

Đây là hàm tiện ích cho giá trị 1 nếu hai vật được nối với nhau qua một khớp có kiểu joint_type, còn không trả về 0. joint_type có giá trị kiểu dJointTypeXXX. Hàm này có hữu ích giúp cân nhắc nên chăng dùng thêm khớp tiếp xúc để nối hai vật. Nếu hai vật thể đã được nối bằng khớp khác khớp tiếp xúc thì việc thêm là không cần thiết. Tuy nhiên nó cho phép thêm nhiều khớp tiếp xúc giữa hai vật kể cả nó đã được nối khớp tiếp xúc.



Hình 7.1 Khớp cầu

7.3 Tham số của khớp

7.3.1 Khớp cầu

Khớp cầu có dạng như **hình 7.1**

```
void dJointSetBallAnchor (dJointID, dReal x, dReal y, dReal z);
```

Đây là hàm đặt tâm của khớp, khớp cố giữ điểm này trên hai vật thể được xác định bằng tọa độ tuyệt đối.

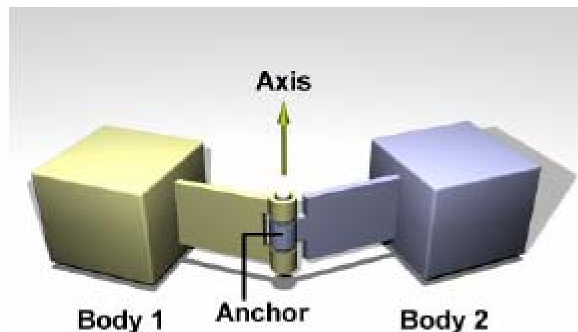
```
void dJointGetBallAnchor (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body1. Nếu hoàn hảo thì điểm này trùng với điểm trên body2.

```
void dJointGetBallAnchor2 (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body2. Bạn có nghĩ là hai hàm `dJointGetBallAnchor()` và `dJointGetBallAnchor2()` cho cùng một kết quả. Nhưng đó là khớp lý tưởng hàm này chỉ cho kết quả giống `dJointGetBallAnchor()` khi mà quá trình quay của khớp không có lỗi. Hàm `dJointGetBallAnchor2()` có thể được dùng cùng với hàm `dJointGetBallAnchor()` để biết điểm tiếp xúc của hai vật thể cách nhau như thế nào.

7.3.2 Khớp bản lề



Hình 7.2: Khớp bản lề

Khớp bản lề có dạng như **hình 7.2**

```
void dJointSetHingeAnchor (dJointID, dReal x, dReal y, dReal z);
void dJointSetHingeAxis (dJointID, dReal x, dReal y, dReal z);
```

Hàm thiết lập tâm khớp và trục quay của khớp

```
void dJointGetHingeAnchor (dJointID, dVector3 result);
```

Hàm lấy giá trị của tâm khớp trong hệ tọa độ toàn cục. Điểm này thuộc body 1. Nếu là khớp lý tưởng thì điểm này cũng thuộc body2.

```
void dJointGetHingeAnchor2 (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body2. Bạn có nghĩ là hai hàm `dJointGetHingAnchor()` và `dJointGetHingAnchor2()` cho cùng một kết quả. Nhưng đó là khớp lý tưởng hàm này chỉ cho kết quả giống `dJointGetHingAnchor()` khi mà quá trình quay của khớp không có lỗi. Hàm `dJointGetHinglAnchor2()` có thể được dùng cùng với hàm `dJointGetHingAnchor()` để biết điểm tiếp xúc của hai vật thể cách nhau như thế nào.

```
void dJointGetHingeAxis (dJointID, dVector3 result);
```

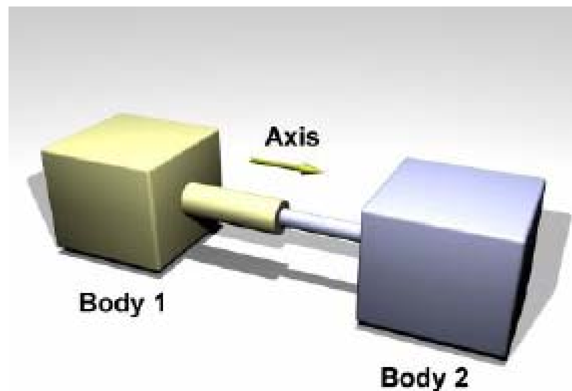
Hàm lấy trục quay của tâm khớp.

```
dReal dJointGetHingeAngle (dJointID);
dReal dJointGetHingeAngleRate (dJointID);
```

Hàm trả về góc quay của khớp và giá trị thời gian, góc được tính là góc giữa hai vật hoặc giữa vật với giá có giá trị trong khoảng –pi đến pi.

Khi thiết lập tâm hoặc trục quay của khớp. Vị trí hiện thời của vật nối với khớp được khảo sát, giá trị đầu của góc được đặt là 0.

7.3.3 Khớp trượt



Hình 7.3 Khớp trượt

```
void dJointSetSliderAxis (dJointID, dReal x, dReal y, dReal z);
```

Thiết lập trục trượt của khớp

```
void dJointGetSliderAxis (dJointID, dVector3 result);
```

Hàm lấy trục trượt của khớp

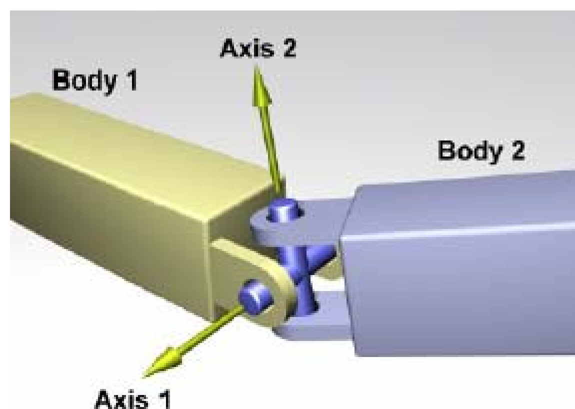
```
dReal dJointGetSliderPosition (dJointID);
dReal dJointGetSliderPositionRate (dJointID);
```

Hàm trả về vị trí của khớp và giá trị thời gian. Khi thiết lập khớp ở trạng thái gắn vật vào thì được tính là vị trí 0.

7.3.4 Khớp Các – đăng

Xem hình 7.4 mô hình khớp Các – đăng

Khớp này khá giống khớp cầu nhưng có một bậc tự. Được thiết lapạ theo kiểu một trục thuộc body1 và một trục thuộc body2. Hai trục này luôn vuông góc với nhau, khi vatạ một quay truyền sang vật hai cũng quay theo.



Hình 7.4 Khớp Các -đăng

Khớp Các –đăng tương đương hai khớp bản lề hai khớp này có trục vuông góc với nhau và được gắn cứng tại tâm của hai khớp.

Sau đây là các hàm về khớp Các – đăng:

```
void dJointSetUniversalAnchor (dJointID, dReal x, dReal y, dReal z);
void dJointSetUniversalAxis1 (dJointID, dReal x, dReal y, dReal z);
void dJointSetUniversalAxis2 (dJointID, dReal x, dReal y, dReal z);
```

Thiết lập tâm khớp và các trục của khớp. Hai trục này phải vuông góc với nhau

```
void dJointGetUniversalAnchor (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body1. Nếu hoàn hảo thì điểm này trùng với điểm trên body2.

```
void dJointGetUniversalAnchor2 (dJointID, dVector3 result);
```

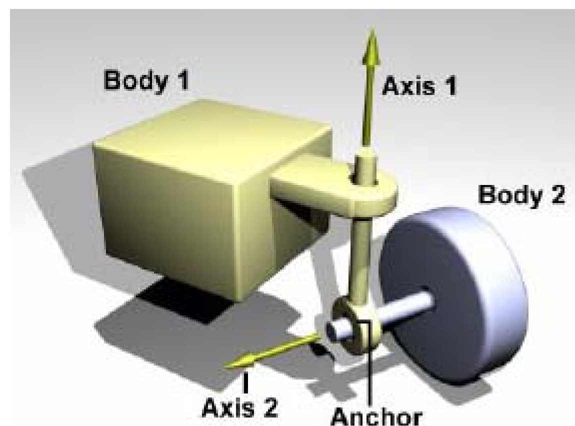
Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body2. Bạn có nghĩ là hai hàm `dJointGetUniversalAnchor ()` và `dJointUniversalAnchor2()` cho cùng một kết quả. Nhưng đó là khớp lý tưởng hàm này chỉ cho kết quả giống `dJointGetHingAnchor()` khi mà quá trình quay của khớp không có lỗi. Hàm `dJointUniversalAnchor2()` có thể được dùng cùng với hàm `dJointGetHingAnchor()` để biết điểm tiếp xúc của hai vật thể cách nhau như thế nào.

```
void dJointGetUniversalAxis1 (dJointID, dVector3 result);
void dJointGetUniversalAxis2 (dJointID, dVector3 result);
```

Hàm lấy trục của khớp.

7.3.5 Khớp quay hai bậc tự do

Xem **hình 7.5** mô hình khớp trượt hai bậc tự do



Hình 7.5 Khớp quay hai bậc tự do

Kết cấu như hai khớp bản lề nối tiếp nhau, có hai trục khác nhau.Ví dụ nhìn vào **hình 7.5** ta thấy đây là cơ cấu ở bánh xe trước điều khiển xe.

Khớp này có một tâm nhưng có hai trục, trục 1 được xác định tương đối với vật 1 (trục này có thể quay nếu vật 1 là khung dầm). Trục 2 được xác định tương đối với vật 2 (trục này có thể là trục quay vật 2 là bánh xe).

Trục 1 có thể quay trong một giới hạn, trục hai chỉ có thể quay tròn.

Trục một có thể như là mọt trục cheo, và chịu nén dọc trục.

```
void dJointSetHinge2Anchor (dJointID, dReal x, dReal y, dReal z);
void dJointSetHinge2Axis1 (dJointID, dReal x, dReal y, dReal z);
void dJointSetHinge2Axis2 (dJointID, dReal x, dReal y, dReal z);
```

Hàm thiết lập tâm khớp và trục quay 1 và trục quay 2. Hai trục này khôgn cùng nằm trên một đường thẳng.

```
void dJointGetHinge2Anchor (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body1. Nếu hoàn hảo thì điểm này trùng với điểm trên body2.

```
void dJointGetHinge2Anchor2 (dJointID, dVector3 result);
```

Hàm lấy tâm khớp có tọa độ tuyệt đối. Điểm này nằm trên body2. Bạn có nghĩ là hai hàm `dJointGetHinge2Anchor ()` và `dJointHinge2Anchor2()` cho cùng một kết quả. Nhưng đó là khớp lý tưởng hàm này chỉ cho kết quả giống `dJointGetHinge2Anchor()` khi mà quá trình quay của khớp không có lỗi. Hàm `dJointHinge2Anchor2()` có thể được dùng cùng với hàm `JointGetHing2Anchor()` để biết điểm tiếp xúc của hai vật thể cách nhau như thế nào.

```
void dJointGetHinge2Axis1 (dJointID, dVector3 result);
void dJointGetHinge2Axis2 (dJointID, dVector3 result);
```

Hàm lấy trục quay

```
dReal dJointGetHinge2Angle1 (dJointID);
dReal dJointGetHinge2Angle1Rate (dJointID);
dReal dJointGetHinge2Angle2Rate (dJointID);
```

Hàm trả về góc quay (quanh trục 1 và quay quanh truc 2) và giá trị thời gian. Khi thiết lập khớp ở trạng thái gắn vật vào thì được tính là vị trí 0.

7.3.6 Khớp cố định

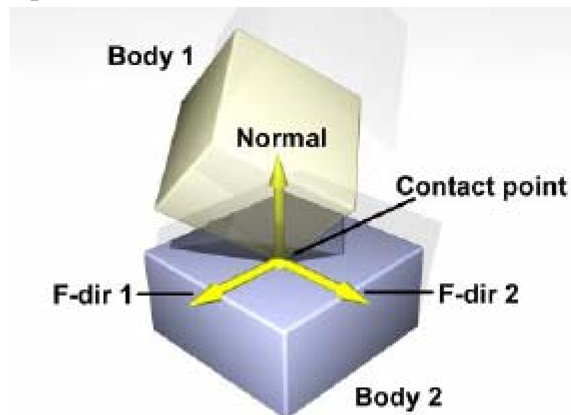
Là khớp thiết lập quan hệ cứng về vị trí, hướng giữa hai vật hoặc giữa vật và giá. Nếu sử dụng khớp này trong khi xây dựng hệ thống mô phỏng thì luôn không là ý tưởng tốt đối với thực tế. Ngoại trừ khi gỡ rối, còn nếu bạn muốn hai vật được gắn cứng với nhau thì tốt hơn là hãy dùng một vật.

```
void dJointSetFixed (dJointID);
```

Gọi hàm làm cứng khớp. Sau khi dùng lệnh này các vị trí tương đối hiện thời giữa hai vật sẽ được lưu và định vị cứng.

7.3.7 Tiếp xúc

Xem **hình 7.6** miêu tả khớp tiếp xúc



Hình 7.6 Khớp tiếp xúc

Khớp tiếp là khớp sinh ra để miêu tả sự tiếp súc giữa hai vật. Làm cho hai vật chuyển động ra sau vơi một vận tốc có hướng và vector pháp tuyến với bề mặt tiếp xúc. Khớp này thông thường sẽ tồn tại ở một bước mô phỏng. Chúng được sinh ra và hủy bỏ ngay khi xuất hiện sự va chạm.

Khớp tiếp xúc có thể mô phỏng được cả ma sát ở vị trí tiếp xúc bởi tính toán có kể đến hai thành phần lực đặc biệt chúng vuông góc với vector pháp tuyến như **hình 7.6**.

Khi khớp tiếp xúc được tạo, cấu trúc dữ liệu dContact sẽ được dùng đến. Kiểu dữ liệu này được định nghĩa như sau:

```
struct dContact {
      dSurfaceParameters surface;
      dContactGeom geom;
      dVector3 fdir1;
};
```

Geom là một cấu trúc dữ liệu con được cung cấp với hàm tìm va chạm, được đề cập đến phần va chạm sau.

fdir1 là thành phần thứ nhất của lực ma sát được định nghĩa là một vector mà dọc theo vector này lực ma sát sẽ tác động vào vật, đây là vector đơn vị và vuông góc với vector pháp tuyến (như là vector tiếp tuyến với bề mặt va chạm). Nó được định nghĩa phụ thuộc vào biến dContactFDir1 là thuộc tính của

surface. Thành phần thứ hai của lực ma sát luôn vuông góc với thành phần thứ nhất và vector pháp tuyến.

Thuộc tính mode của surface luôn luôn phải được thiết lập. Nó là sự kết hợp giữa một hoặc nhiều trạng thái.

dContactMu2	Nếu không set, dùng mu cho cả hai thành phần ma sát. Nếu set, mu dùng cho thành phần thứ nhất, mu2 dùng cho thành phần thứ hai.
dContactFDir1	Nếu dùng, Fdir1 dùng cho thành phần lực ma sát thứ nhất, ngược lại tự động đặt thành phần thứ nhất vuông góc với vector pháp tuyến (trong trường hợp này hướng không thể đoán trước)
dContactBounce	Nếu dùng thì mặt tiếp xúc sẽ là mặt đàn hồi, trong trường hợp còn lại hai vật sẽ không va chạn đàn hồi. Độ chính xác phụ thuộc vào tham số đàn hồi.
dContactSoftERP	Nếu dùng, tham số giảm lỗi của khớp sẽ được thiết lập bởi biến soft_erp. Rất có hiệu ích khi bề mặt va chạm là bề mặt mềm.
dContactSoftCFM	Nếu dùng, tham số CFM của khớp sẽ được thiết lập bởi biến soft_cfm. Rất có hiệu ích khi bề mặt va chạm là bề mặt mềm.
dContactMotion1	Nếu được dùng, thì mặt tiếp xúc được giả thiết dịch chuyển độc lập với dịch chuyển của vật, giống như một băng tải chạy qua bề mặt vật. motion1 được định nghĩa là chuyển động của mặt theo hướng thành phần thứ nhất của lực ma sát
dContactMotion2	Nếu được dùng, thì mặt tiếp xúc được giả thiết dịch chuyển độc lập với dịch chuyển của vật, giống như một băng tải chạy qua bề mặt vật. Motion2 được định nghĩa là chuyển động của mặt theo hướng thành phần thứ hai của lực ma sát
dContactSlip1	Lực (FDS) trên hướng thành phần thứ nhất của lực ma sát
dContactSlip2	Lực (FDS) trên hướng thành phần thứ hai của lực ma sát

dContactApprox1_1	Dùng hình chóp để xấp xỉ cho hướng ma sát 1. Nếu không được xác định rõ thì hằng số lực không đổi được sử dụng (Mu là lực giới hạn)
dContactApprox1_2	Dùng hình chóp để xấp xỉ cho hướng ma sát 2. Nếu không được xác định rõ thì hằng số lực không đổi dược sử dụng (Mu là lực giới hạn)
dContactApprox1	Tương đương dùng cả dContactApprox1 1 và dContactApprox1 2

dReal mu: Hệ số ma sát Culông, trong khoảng 0 ÷ ∞. Nếu bằng 0 tiếp xúc không ma sát, ∞ là tiếp xúc không trượt. Tham số này luôn phải được thiết lập.

dReal mu2: Hệ số ma sát Culông cho thành phần 2, trong khoảng 0 ÷ ∞. Nếu bằng 0 tiếp xúc không ma sát, ∞ là tiếp xúc không trượt.

dReal bounce: Tham số đàn hồi (0..1). Nếu bằng 0 là bề mặt không đàn hồi, 1 là bề mặt đàn hồi tuyệt đối.). Các tham số này chỉ được thiết lập khi thiết lập biến mode.

dReal bounce_vel: Vận tốc đầu vào tối thiểu cho va chạm đàn hồi (m/s). Vận tốc tối thiểu đầu vào sẽ có ảnh hưởng đến tham số đàn hồi khi bằng 0.). Các tham số này chỉ được thiết lập khi thiết lập biến mode.

dReal soft_erp: Tham số điều khiển độ mềm va chạm.). Các tham số này chỉ được thiết lập khi thiết lập biến mode.

dReal soft_cfm: Tham số điều khiển độ mềm va chạm. Khi). Các tham số này chỉ được thiết lập khi thiết lập biến mode.

dReal motion1, motion2: Vận tốc bề bặt theo phương ma sát 1 và 2 (m/s). Các tham số này chỉ được thiết lập khi thiết lập biến mode.

dReal slip1, slip2: Các hệ (FDS) số lực phụ thuộc vào các lực ma sát theo các hướng 1 và hướng 2. Các tham số này chỉ được thiết lập khi thiết lập biến mode.

FDS là một hiệu ứng gây ra do mặt tiếp xúc của các mặt bên với nhau ở vận tốc mà lực này cân đối với lực tác động tiếp tuyến vào vật.

Khi xem xét một điểm tiếp xúc có hệ số ma sát là μ bằng vô cùng. Bình thường, nếu một lực f tác động vào 2 mặt tiếp xúc. Hãy cho chúng trượt qua lại lẫn nhau. Chúng sẽ không dịch chuyển. Tuy nhiên, nếu hệ số FDS được thiết lập với giá trị dương k thì hai bề mặt sẽ dịch chuyển qua lại nhau. Theo quan hệ k*f.

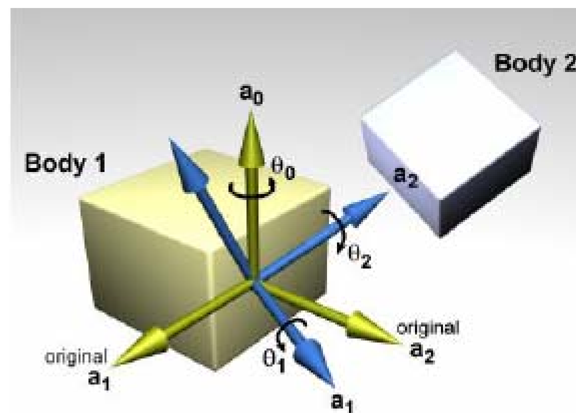
Chú ý: Cái này không giống các kiểu ma sát thông thường, lực này không sinh ra khi gia tốc dịch chuyển giữa các vật không đổi .

Điều này rất quan trọng đối với mô hình có nhiều trạng thái, đặc biệt là ở cơ cấu vận động. Ví dụ Một chiếc ô tô chuyển động trên đường. Đẩy chuyển động thẳng hướng (các bộ phận vận động bắt đầu quay). Đẩy xe theo phương vuông góc xe sẽ không chuyển động (các bộ phận vận động không quay). Tuy nhiên., nếu xe chuyển động với vận tốc *v*, cho một lực *f* tác động vuông góc với phương chuyển động vào xe gây ra các chuyển động trượt ở bánh xe với vận tốc *f**v* (điều này luôn xảy ra).

Mô hình này trong thư viện thiết lập các tham số cho các chuyển động bánh xe như sau: Thiết lập thnàh phàn thứ nhất lực ma sát theo phương mà bánh xe quay, và đặt hệ số FDS theo phươgn của thành phần thứ hai của lực ma sát là *k**v*, trong đó v vận tốc quay của bánh xe, *k* hệ số phụ thuộc vào bánh xe chọn theo kinh nghiệm.

Chú ý: FDS được dùng riêng biệt ở cả trường hợp stick/trượt theo công thực Culông – cả hai mô hình được sử dụng cùng nhau tại các điểm tiếp xúc đơn.

7.3.8 Khớp quay (AMotor)



Hình 7.7 Khớp quay với góc quay euler

Khớp quay cho phép hai vật quay tương đối với nhau. Góc quay có thể điều khiển phụ thuộc vào 3 trục. Cho phép đặt các mô mem chủ động và mô mem dừng theo các trục (xem ở phần đặt tham số mô mem cho khớp quay) Điều này hữu ích cho khớp cầu (không thiết lập rằng buộc góc cho tất cả các bậc tự do). Có thể thiết lập rằng buộc cho bất kỳ trục quay nào là cần thiết. Khi dùng khớp quay như khớp cầu, khớp cũng được nối với hai vật như ở khớp cầu.

Khớp quay có thể dùng được ở nhiều kiểu khác nhau. Ở chế độ dAMotorUser, người dùng đặt các trục quay để điều khiển, ở chế độ dAMotorEuler, khớp tính toán góc Euler tương ứng với góc quay, cho phép dùng góc Euler để đặt mô mem lực. Khớp quay ở chế độ dAMotorEuler như ở **hình 7.7**.

Trong hình, a_0, a_1 và a_2 là 3 trục của góc Euler điều khiển, trục mầu xanh *(a_0)* có gốc đặt trên body1, trục mầu xanh nước biển *(a_2)* gốc đặt trên body2. Khi lấy hệ tọa độ của body2 theo hệ tọa độ của body1 cần thực hiện các bước biến đổi sau:

- Quay một góc θ_0 quay trục a_0
- Quay một góc θ_1 quay trục a_1 (a_1 được quay theo tọa độ gốc)
- Quay một góc θ_2 quay trục a_2 (a_2 được quay hai lần theo tọa độ gốc)

Có một hạn chế rất quan trọng khi sử dụng khớp quay ở chế độ góc Euler là: θ_1 không được vượt quá phạm vi cho phép $-\pi/2 \div \pi/2$ nếu điều này vi phạm thì khớp quay sẽ không ổn định (đặc biệt ở vùng $+/\div-\pi/2$) như vậy bạn phải cho trục 1 dừng.

```
void dJointSetAMotorMode (dJointID, int mode);
int dJointGetAMotorMode (dJointID);
```

Đặt và lấy kiểu khớp quay. Kiểu khớp có dạng một trong các kiểu sau:

dAMotorUser	Trục quay và góc quay được người dùng thiết lập. Đây là kiểu mạc định.
dAMotorEuler	Góc Euler được tự động tính toán, trục a_1 được cho trước, các trục còn lại phải được đặt đúng như miêu tả phía sau. Khi khớp quay được khởi tọa ở dạng này các quan hệ về hướng của vật phụ thuộc vào góc Euler 0

```
void dJointSetAMotorNumAxes (dJointID, int num);
int dJointGetAMotorNumAxes (dJointID);
```

Hàm đặt/lấy số hiệu trục quay để điều khiển khớp. num có giá trị trong khoảng 0 – 3. Ở đây tự động đặt num =3 là khớp ở dạng dAMotorEuler.

```
void dJointSetAMotorAxis (dJointID, int anum, int rel,
                          dReal x, dReal y, dReal z);
void dJointGetAMotorAxis (dJointID, int anum, dVector3 result);
int dJointGetAMotorAxisRel (dJointID, int anum);
```

Hàm đặt/lấy trục quay của khớp, tham số anum có giá trị là 0, 1 hoặc 2 mỗi trục của khớp có thể là một trong các trường hợp rel có giá trị sau:

- 0: Tâm của trục tọa độ trung tâm của hệ tọa độ thế giới.
- 1: Tâm của trục tọa độ nằm trên body1.
- 2: Tâm của trục tọa độ nằm trên body2.

Vector miêu tả trục tọa độ (x,y,z) luôn luôn được chỉ rõ trong hệ tọa độ toàn cầu bất chấp các giá trị của rel. Có hai hàm GetAMotorAxis, một trả về trục tọa độ và một trả về quan hệ với kiểu khớp.

Đối với kiểu Euler:

- Chỉ cần đặt trục 0 và trục 2, trục một sẽ được tự động tính toán ngay ở bước mô phỏng sau.
- Trục 0 và trục 2 phải vuôgn góc với nhau
- Trục 0 có gốc đặt trên body1, trục 2 có gốc trên body2

```
void dJointSetAMotorAngle (dJointID, int anum, dReal angle);
```

Cho biết giá trị góc quay quanh trục anum hiện thời là bao nhiêu. Hàm này chỉ được dùng ở chế độ dAMotorUser. Bởi vì ở chế độ này khớp không tự xác định được góc quay hiện thời, thông tin này là cần thiết khi không cho khớp quay quanh một trục nào đó, và không cần thiết cho trục đạng quay.

```
dReal dJointGetAMotorAngle (dJointID, int anum);
```

Cho biết giá trị quay của góc quanh trục anum. Trong chế độ dAMotorUser đơn giả giá trị này là tập hợp các giá trị mà được hàm `dJointSetAMotorAngle()` thiết lập. Ở chế độ dAMotorEuler nó phụ thuộc vào góc Euler.

```
dReal dJointGetAMotorAngleRate (dJointID, int anum);
```

Return the current angle rate for axis anum. Ở chế độ dAMotorUser hàm này luôn là 0. Trong chế độ In dAMotorEuler nó phụ thuộc vào rate của góc Euler.

7.4 Tham số chung

Các hàm thiết lập các tham số hình học của khớp được gọi sau khi các khớp nối với vật và các vật này đã được định vị chính xác, nếu không chúng không được khởi tạo chính xác. Nếu các khớp khôgn được nối với vật thì các hàm này sẽ không có tác động nào.

Các hàm trả về các giá trị của tham số này, khi hệ thống có lỗi (có một vài khớp có lỗi) tâm và trục của khớp chỉ nằm trên body1 (hoặc body2 nếu một trong hai là giá) .

- Giá trị mạc định của tâm là (0,0,0) và của trục khớp là (1,0,0)

- Khi một trục được khởi tạo sẽ được chuyển thành vector đơn vị. Trục nào được khởi tọa hàm trả về sẽ có giá trị.
- Khi đo đạc góc hoặc các vị trí, các giá trị này phụ thuộc vào giá trị khởi tạo quan hệ ban đầu của các vật.

Chú ý: Không có hàm thiết lập trực tiếp góc và vị trí của khớp, thay vào đó bạn phải đặt vị trí và vận tốc của vật.

7.5 Dừng khớp và tham số của động cơ

Khi khớp được tạo ra không có gì ngăn cản nó chuyển động xuyên xuốt phạm vi chuyển động của nó. Ví dụ khớp bản lề có thể quay tới bất kì góc nào, khớp trượt có thể trượt tới bất kì đâu.

Khoảng làm việc của khớp có thể được hạn chế bằng cách thiết lập vị trí giới hạn, Góc quay (vị trí) của khớp có thể được hạn chế tại cận làm việc dưới, hoặc tại cận làm việc trên. Chú ý là góc (vị trí) ban đầu của khớp (giá trị 0) phụ thuộc vào vị trí ban đầu của các vật nối với khớp.

Khi hạn chế khoảng làm việc của khớp, cách tốt nhất là sử dụng động cơ để sinh lực cản tác động vào khớp làm cho chuyển động của vật đạt được tốc độ mong muốn. Nhưng lực này không thể sinh lực vượt qúa lực/mômen cho phép tác động vào khớp.

Động cơ có hai tham số: Tốc độ mong muốn và lực tác động để có tốc độ mong muốn đó. Đây là mô hình đơn giản trong thực tế, động cơ hay servo. Nhưng rất hữu ích khi sử dụng mô hình này cùng với hộp số để có được nhiều tỉ số truyền trước khi dùng để điều khiển tốc độ làm việc cảu khớp.

Một cách đơn giản, có thể thay thế động cơ bằng cách tác động trực tiếp một ngoại lực vào vật. Tuy nhiên lực tác động trực tiếp thường không phải là cách tiếp cận tốt và có thể dẫn đến những vấn đề về ổn định. Khi đặt lực tác động vào vật để thu được tốc độ mong muốn, ta có thể tính toán lực dựa vào các thông tin về vận tốc hiện thời, theo công thức sau:

Lực = k(Vận tốc mong muốn – vận tốc hiện thời)* *(7.1)*

Có vài vấn đề ở đây. Thứ nhất là hệ số *k* phải được điều chỉnh bằng tay . Nếu *k* quá nhỏ vật phải mất nhiều thời gian để đạt được vận tốc mong muốn, nếu *k* quá lớn thì hệ thống không ổn định. Thứ hai là ngay cả khi *k* được chọn hợp lý thì vật cũng cần một số bước để đạt tới vận tốc mong muốn . Thứ ba là nếu có bất kì lực ngoài nào tác động vào vật, thì vận tốc mong muốn có thể, thậm chí không bao giờ đạt được.

Động cơ giải quyết tất cả các vấn đề trên, chúng đưa vật đạt đến vận tốc mong muốn trong một khoảng thời gian, động cơ không cần nhiều tham số bởi vì chúng thực sự làm việc như những rằng buộc. Điều này làm cho lực động cơ được sử dụng nhiều hơn là đặt lực trực tiếp vào vật, và làm cho hệ thống bền vững và ổn định hơn, và cần ít thời gian để thiết kế hệ thống. Đặc biệt với những cơ hệ lớn.

7.5.1 Hàm đặt tham số

Đây là các hàm thiết lập các tham số về cận làm việc của khớp và các hàm về động cơ.

```
void dJointSetHingeParam (dJointID, int parameter, dReal value);
void dJointSetSliderParam (dJointID, int parameter, dReal value);
void dJointSetHinge2Param (dJointID, int parameter, dReal value);
void dJointSetUniversalParam (dJointID, int parameter, dReal value);
void dJointSetAMotorParam (dJointID, int parameter, dReal value);
dReal dJointGetHingeParam (dJointID, int parameter);
dReal dJointGetSliderParam (dJointID, int parameter);
dReal dJointGetHinge2Param (dJointID, int parameter);
dReal dJointGetUniversalParam (dJointID, int parameter);
dReal dJointGetAMotorParam (dJointID, int parameter);
```

Tham số `parameter` có thể nhận các giá trị sau:

dParamLoStop	Đặt cận làm việc dưới của khớp, giá trị của biến dInfinity được đặt mạc định để tắt hạn chế dưới. Với khớp quay cận dưới phải lớn hơn $-\pi$ mới có tác dụng.
dParamHiStop	Đặt cận làm việc trên của khớp, giá trị của biến dInfinity được đặt mạc định để tắt hạn chế trên. Với khớp quay cận trên phải lớn hơn π mới có tác dụng. Nếu đặt cận trên nhỏ hơn cận dưới thì cả hai đều không có tác dụng.
dParamVel	Đặt tốc độ mong muốn của động cơ (có thể là vanạ tốc thẳng hay vận tốc góc)
dParamFMax	Đặt lực/mômen cực đại tương ứng vơi tốc độ cảu động cơ. Luôn phải đặt >=0, giá trị 0 để tắt động cơ đây là giá trị mạc định

dParamFudgeFactor	Khi khớp đang ở vị trí giới hạn mà động cơ vẫn làm việc nếu lực quá lớn sẽ dẫn đến phá vỡ khớp. Hệ số Fudge được sử dụng để khắc phục điều này, nó nhận giá trị từ 0 ÷ 1 (1 là giá trị mạc định). Nếu nguy cơ vỡ khớp xảy ra giá trị này dần giảm xuống đủ nhỏ để ngăn chặn lực gây vỡ khớp
dParamBounce	Đây là tham số có giá trị 0 ÷ 1, giá trị 0 là tại các vị trí giới hạn khớp không đàn hồi, giá trị 1 là tại vị trí này có tính đàn hồi tuyệt đối.
dParamCFM	Lực rằng buộc hỗn hợp (CFM) được sử dụng tại các vị trí không phải là vị trí giới hạn của khớp.
dParamStopERP	Tham số giảm sai số (ERP) được sử dụng tại vị trí giới hạn của khớp.
dParamStopCFM	Lực rằng buộc hỗn hợp (CFM) được sử dụng tại các vị trí giới hạn của khớp.
dParamSuspensionERP	Tham số giảm sai số (ERP) ở các khớp giảm chấn. Hiện tại mới có ở khớp bản lề hai bậc tự do.
dParamSuspensionCFM	Lực rằng buộc hỗn hợp (CFM) ở các khớp giảm chấn. Hiện tại mới có ở khớp bản lề hai bậc tự do.

Tên của các tham số parameter có thể được thêm vào sau các chữ số (2 hoặc 3) để chỉ ra rằng các giá trị thứ 2 hoặc thứ 3 của tham số parameter được thiết lập. Như trục thứ hai của khớp bản lề hai bậc tự do, hoặc trục thứ 3 của khớp quay. Một tham số cố định dParamGroup được định nghĩa theo công thức sau dParamXi = dParamX + dParamGroup *(i – 1)

7.6 Đặt lực/ Mô mem xoắn trục tiếp

Động cơ được sử dụng để điều khiển tốc độ của khớp, tuy nhiên ta có một cách khác để làm điều này đó là dùng trực tiếp các lực/mômen tác động vào vật. Chú ý là không hiệu quả bằng dùng động cơ, khi dùng đơn gian ta chỉ cần gọi các hàm `dBodyAddForce()/dBodyAddTorque()` đặt vào vật.

`dJointAddHingeTorque(dJointID joint, dReal torque)`

Hàm này đặt mômen torque quanh trục của khớp bản lề, tác động vào body1 và tác động vào body2 theo chiều ngược lại, đây là một hàm của gói `dBodyAddTorque().`

`dJointAddUniversalTorques(dJointID joint, dReal torque1, dReal torque2)`

Hàm đặt mômen torque1 quanh trục một của khớp Cac-đăng, và mômen torque2 quanh trục hai, đây là một hàm của gói `dBodyAddTorque().`

`dJointAddSliderForce(dJointID joint, dReal force)`

Hàm đặt lực theo trục của khớp trượt. Tác động vào body1 và tác động vào body2 theo chiều ngược lại, đây là một hàm của gói `dBodyAddForce().`

`dJointAddHinge2Torqe(dJointID joint, dReal torque1, dReal torque2)`

Hàm đặt mômen torque1 quanh trục 1 của khớp bản lề hai bậc tự do, và mômen torque2 quanh trục 2, đây là một hàm của gói dBodyAddTorque().

```
dJointAddAMotorTorques(dJointID joint, dReal torque0, dReal torque1,
                       dReal torque2)
```

Hàm đặt mômen `torque0` quanh trục 0 của khớp quay, mômen torque1 quanh trục 1 và mômen torque2 quanh trục 2 đây là một hàm của gói `dBodyAddTorque().`

Chương 8

STEPFAST

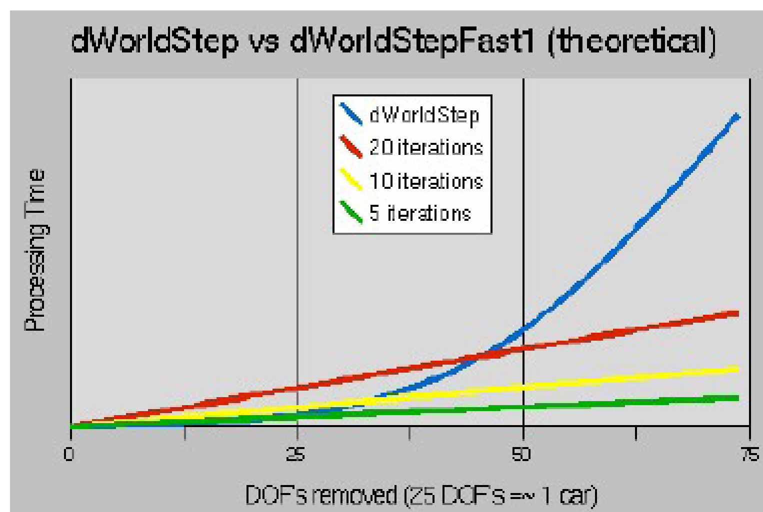
Hàm dWorldStep() trong thư viện sử dụng phương phát ma trận lớn cho mỗi bước mô phỏng. Nếu hệ thống mô phỏng lớn tốc độ mô phỏng sẽ chậm và cần nhiều bộ nhớ cho việc tính toán. Giải thuật StepFast1 được sử dụng để thay thế tính toán trên mỗi bước mô phỏng, đổi lại một ít độ chính xác để lấy tốc độ và giảm bộ nhớ. Để sử dụng ta chỉ cần gọi hàm dWorldStepFast1() thay vì dWorldStep().

Hình 8.1 minh họa lợi thế tốc độ qua giải thuật dWorldStep chuẩn.

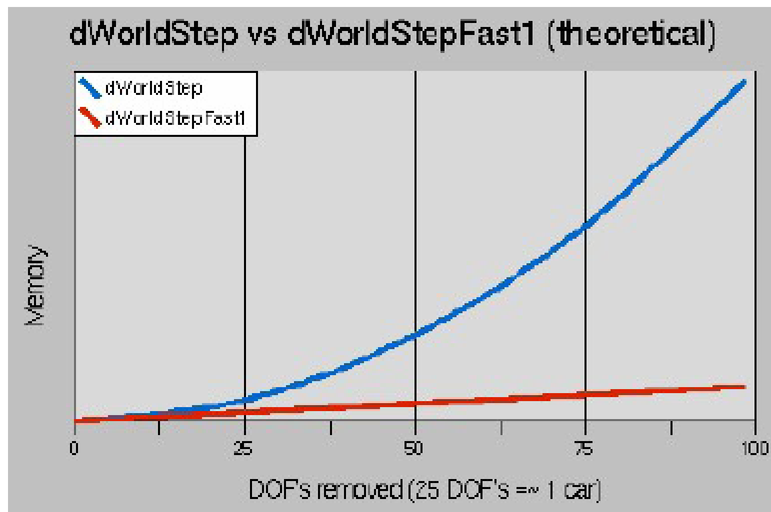
Đồ thị thể hiện mối quan hệ giữa số bậc tự do trong hệ thống mô phỏng và thời gian mỗi bước tính toán. Thấy ngay ở giải thuật dWorldStep() thời gian mỗi bước tính là một hàm bậc ba của số bậc tự do của hệ thống. Nhưng ở giải thuật thay thế dWorldStepFast1() đây là hàm bậc nhất.

Hình 8.2 minh họa lợi thếvề bộ nhớ qua giải thuật dWorldStep chuẩn.

Đồ thị thể hiện mối quan hệ giữa số bậc tự do trong hệ thống mô phỏng và dung lượng bộ nhớ. Thấy ngay ở giải thuật dWorldStep()dung lượng bộ nhớ là một hàm bậc hai của số bậc tự do của hệ thống. Nhưng ở giải thuật thay thế dWorldStepFast1() đây vẫn là hàm bậc nhất.



Hình 8.1 lợi thế tốc độ của giải thuật StepFast1



Hình 8.2 lợi thế về bộ nhớ của giải thuật StepFast

8.1 Khi nào dùng StepFast1

Ở các hình trên cho ta biết giải thuật StepFast khá tốt để tăng tốc độ và giảm bộ nhớ, nhưng tất cả các sức mạnh đó không phải đến tự do, tất cả các tối ưu hóa cần có sự cân bằng theo các nguyên tắc. Giải thuật StepFast đổi độ chính xác mô phỏng lấy tốc độ và lợi thế bộ nhớ. Như vậy bạn cần phải chọn độ chính xác như thế nào là có thể chấp nhận được mà vẫn bảo đảm được tốc độ mô phỏng.

Câu hỏi này sẽ được trả lời như sau: Khi sử dụng StepFast mà không quan tâm đến một số tham số có ảnh hưởng đến độ ổn định của hệ thống, đổi lại bạn cần có lợi thế về tốc độ và bộ nhớ. Nếu trong khi mô phỏng bạn tìm được một trạng thái mà tại đó có số lượng lớn vật tiếp xúc với nhau thì giải thuật dWorldStep() sẽ chậm cần phải chuyển trang giải thuật StepFast. Một số hệ thống làm việc tốt khi đổi hàm dWorldStep() thành hàm dWorldStepFast(), một số khác cần có một khoảng thời gian ngừng để chuyển trạng thái, ở đây khối lượng của vật luôn luôn gây ảnh hưởng đặt biệt là khớp nối mà vật có khối lượng chênh nhau lớn. Giải thuật StepFast có thể gặp những rắc rối lớn khi giải quyết.

Khi dùng giải thuật StepFast để thiết kế hệ thống có tính đến địa hình, bạn phải biết rằng sẽ có rất nhiều vật trên đó, bạn cần lập phương án sử dụng StepFast nhưng ở đây có vấn đề về tỉ lệ khối lượng giữa các vật. Bạn phải có cách nào đó làm cho tỉ lệ khối lượng của các vật gần bằng 1 hoặc nằm trong một khoảng nhỏ. Ví dụ nằm trong khoảng $0.5 \div 1.5$. Một điều cuối cùng là bạn chỉ sử dụng giải thuật StepFast khi nào mà thực sự cần tới.

8.2 Khi nào không dùng StepFast1

Tuy nhiên có vài trạng thái đặc biệt, khi đó việc sử dụng giải thuật StepFast1 không còn thích hợp, Tôi tin tởng rằng một vài trường hợp sau là đúng. Không sử dụng StepFast1 khi sự chính xác quan trọng hơn tốc độ hoặc bộ nhớ.

8.4 Các hàm tiện tích đi cùng StepFast1

Có một số hàm được thêm vào trong thư viện có liên quan đến giải thuật StepFast, và chỉ có tác dụng khi hệ thống sử dụng giải thuật này một số hàm sẽ tự động phụ thuộc vào trrạng thái của vật (Được kích hoạt, hay không được kích hoạt) một số khác cần được tối hưu hóa. Dưới đây là một số ý kiến sử dụng chúng:

- The body is considered a candidate for disabling when it falls below a certain speed (linear and angular), called the AutoDisableThreshold. In the interest of speedy execution, the actual speed measured is the square of the speed of the body. So you may need to set a lower value than you expected. 0.003 works well in test crash, and is the default.

- When the body has remained a disable candidate for a certain number of steps (AutoDisableSteps), it

is disabled. This is almost completely for boxes, which like to land and bounce up on two points, and teeter motionless for a few steps before falling back down. Round items generally need a much lower (like 1) AutoDisableSteps than boxes do (10+), 10 is the default.

- AutoDisabling is disabled by default, use dBodySetAutoDisableSF1(body, true) to enable it.

- A body is automatically re-enabled when it comes in contact with another enabled body.

- Enabled bodies only enable bodies within (AutoEnableDepth) bodies of them each step. This, in conjunction with AutoDisabling, causes a rim of bodies that are enabled and disabled each step to form, containing the enabled bodies to the smallest area allowed by the AutoDisable parameters. Setting AutoEnableDepth to a really large number will retain the current functionality. Setting it to 0

will give you a new functionality: disabled bodies will never be automatically re-enabled, acting like geoms only. 3 seems to be a good value for the wall in test crash, but 1000 is the default to retain standard functionality.

Note that the functions pertaining to auto-disabling are not yet implemented!

8.5 API

```
void dWorldStepFast1(dWorldID, dReal stepsize, int maxiterations);
```

Bước tính được cho vào theo tham biến stepsize (tính bằng giây) được sử dụng ở giải thuật StepFast1. Số vòng lặp dược thực hiện được đưa vào theo tham biến maxiterations.

```
void dWorldSetAutoEnableDepthSF1(dWorldID, int autoEnableDepth);
int dWorldGetAutoEnableDepthSF1(dWorldID);
```

Hàm đặt và láy giá trị autoEnableDepth trong giải thuật StepFast1.

```
void dBodySetAutoDisableThresholdSF1(dBodyID, dReal autoDisableThreshold);
dReal dBodyGetAutoDisableThresholdSF1(dBodyID);
```

Hàm đặt và lấy (cho vật) AutoDisableThreshold trong giải thuật StepFast1.

```
void dBodySetAutoDisableStepsSF1(dBodyID, int AutoDisableSteps);
int dBodyGetAutoDisableStepsSF1(dBodyID);
```

Hàm đặt và lấy (cho vật) AutoDisableStep trong giải thuật StepFast1.

```
void dBodySetAutoDisableSF1(dBodyID, int doAutoDisable);
int dBodyGetAutoDisableSF1(dBodyID);
```

Hàm đặt và lấy (cho vật) trạng thái của biến AutoDisable trong giải thuật StepFast1. Nếu biến `doAutoDisable` khác 0, AutoDisable sẽ hiệu lực. Nếu biến `doAutoDisable` bằng 0, AutoDisable sẽ không có hiệu lực.

Chương 9

HÀM TRỢ GIÚP

9.1 Hàm thiết lập góc quay

Hướng của vật được miêu tả bởi quaternion gồm 4 phần tử [q0,q1,q2,q3]. Với mỗi vât có ma trận quay 3x3 mà được dẫn xuất ra từ quaternion và hai kiểu thong tin này luôn cho thông tin trùng khớp nhau.

Một số thông tin về quaternion:

- q và –q luôn cho cùng một hướng
- nghịch đảo của q là [q[0]-q[1]-q[2]-q[3]]

Dưới đây là một số hàm tiện ích về ma trận quay và quaternion .

```
void dRSetIdentity (dMatrix3 R);
```

Chuyển ma trận R về ma trận đơn vị .

```
void dRFromAxisAndAngle (dMatrix3 R,
                         dReal ax, dReal ay, dReal az, dReal angle);
```

Tạo ma trận R quanh trục (ax,ay,az) một góc angle (radian).

```
void dRFromEulerAngles (dMatrix3 R,
                        dReal phi, dReal theta, dReal psi);
```

Tạo ma trận R từ ba góc Euler.

```
void dRFrom2Axes (dMatrix3 R, dReal ax, dReal ay, dReal az,
                  dReal bx, dReal by, dReal bz);
```

Tạo ma trận quay từ hai vector 'a' `(ax,ay,az)`, 'b' `(bx,by,bz)`. Sao cho 'a', 'b' là hai trục x, y của hệ tọa độ. Chú ý 'a', 'b' là hai vector đơn vị và 'a' phải vuông góc với 'b'.

```
void dQSetIdentity (dQuaternion q);
```

Chuyển `q` về ma trận đơn vị .

```
void dQFromAxisAndAngle (dQuaternion q, dReal ax, dReal ay, dReal az,
                         dReal angle);
```

Tạo `q` quanh trục `(ax,ay,az)` một góc angle (radian).

```
void dQMultiply0 (dQuaternion qa,
```

```
                   const dQuaternion qb, const dQuaternion qc);
void dQMultiply1 (dQuaternion qa,
                   const dQuaternion qb, const dQuaternion qc);
void dQMultiply2 (dQuaternion qa,
                   const dQuaternion qb, const dQuaternion qc);
void dQMultiply3 (dQuaternion qa,
                   const dQuaternion qb, const dQuaternion qc);
```

Nhân `qa=qb*qc`, giống như `qa=` quay `qc` quanh qb. `dQMultiply1` dùng nghịch đảo của `qb`, `dQMultiply21` dùng nghịch đảo của `qc`, `dQMultiply3` dùng nghịch đảo của cả hai `qb, qc`.

```
void dQtoR (const dQuaternion q, dMatrix3 R);
void dRfromQ (dMatrix3 R, const dQuaternion q);
```

Chuyển `q` thành ma trận `r`.

```
void dRtoQ (const dMatrix3 R, dQuaternion q);
void dQfromR (dQuaternion q, const dMatrix3 R) ;
```

Chuyển ma trận `r` thành `q`.

```
void dWtoDQ (const dVector3 w, const dQuaternion q, dVector4 dq);
void dDQfromW (dReal dq[4], const dVector3 w, const dQuaternion q);
```

Nhân `q` với vector vận tốc góc `w` trả về dq =dq/dt.

9.2 Hàm thiết lập khối lượng

Khối lượng của vật được định nghĩa theo cấu trúc dMass như sau:

```
typedef struct dMass {
      dReal mass; // T•ng kh•i l••ng v•t
      dVector4 c; // T•a •• tr•ng tâm (x,y,z) theo t•a •• v•t
      dMatrix3 I; //  ma tr•n quán tính 3x3
} dMass;
```

Các hàm có liên quan như sau:

```
      void dMassSetZero (dMass *);
```

Đặt toàn bộ khối lượng vật về bằng 0.

```
void dMassSetParameters (dMass *, dReal themass,
                         dReal cgx, dReal cgy, dReal cgz,
```

```
                              dReal I11, dReal I22, dReal I33,
                              dReal I12, dReal I13, dReal I23);
```

Đặt toàn bộ khối lượng vật về bằng =themass. Có tọa độ trọng tâm tại. (cgx,cgy,cgz) theo tọa độ vật. Các tham số Ixx là phần tử của ma trận quán tính của vật:

```
[ I11 I12 I13 ]
[ I12 I22 I23 ]
[ I13 I23 I33 ]
void dMassSetSphere (dMass *, dReal density, dReal radius);
void dMassSetSphereTotal (dMass *, dReal total_mass, dReal radius);
```

Đặt khối lượng của vật cầu có bán kính là radius và [bật độ phân bố bằng density hoặc tổng khối lượng bằng total_mass], tọa độ trọng tâm tại (0,0,0).

```
void dMassSetCappedCylinder (dMass *, dReal density, int direction,
                             dReal radius, dReal length);
void dMassSetCappedCylinderTotal (dMass *, dReal total_mass,
                             int direction, dReal radius, dReal length);
```

Đặt khối lượng của vật trụ có mũ bán cầu "capped cylinder", bán kính là radius và [bật độ phân bố bằng density hoặc tổng khối lượng bằng total_mass], có độ dài là length (không tính phần mũ bán cầu), dọc theo trục tọa độ direction của vật (1=x, 2=y, 3=z), tọa độ trọng tâm tại (0,0,0).

```
void dMassSetCylinder (dMass *, dReal density, int direction,
                       dReal radius, dReal length);
void dMassSetCylinderTotal (dMass *, dReal total_mass, int direction,
                       dReal radius, dReal length);
```

Đặt khối lượng của vật trụ có bán kính là `radius` và [bật độ phân bố bằng density hoặc tổng khối lượng bằng total_mass], có độ dài là `length` dọc theo trục tọa độ `direction` của vật (1=x, 2=y, 3=z), tọa độ trọng tâm tại (0,0,0).

```
void dMassSetBox (dMass *, dReal density,
                  dReal lx, dReal ly, dReal lz);
void dMassSetBoxTotal (dMass *, dReal total_mass,
                  dReal lx, dReal ly, dReal lz);
```

Đặt khối lượng của vật hình hộp có độ dài các cạnh là lx, ly, lz và [bật độ phân bố bằng density hoặc tổng khối lượng bằng total_mass], tọa độ trọng tâm tại (0,0,0).

```
void dMassAdjust (dMass *, dReal newmass);
```

Đặt khôi lượng cho vật bằng tổng khối lượng mới.

```
void dMassTranslate (dMass *, dReal x, dReal y, dReal z);
```

Chuyển tọa độ trọng tâm của vật đến vị trí `(x,y,z)`.

```
void dMassRotate (dMass *, const dMatrix3 R);
```

Lấy thông số khối lượng của vật bằng cách quay chúng bởi ma trận `R`.

```
void dMassAdd (dMass *a, const dMass *b);
```

Thêm khối lượng `b` vào khối lượng `a`.

9.3 Hàm tính toán toán học

Tiếp tục cần bổ xung

9.4 Lỗi và các hàm về bộ nhớ

Chương 10

DÒ TÌM VA CHẠM

Thư viện gồm hai thành phần chính. Một là mô phỏng động lực học, hai là tìm va chạm. Thành phần tìm va chạm lấy thông tin hình học của vật. Tại mỗi bước mô phỏng nó kiểm tra sự va chạm lẫn nhau cảu các vật thể và phải hồi lại các điểm tiếp xúc cho người dùng. Người dùng tạo ra các khớp tiếp xúc giữa các vật chạm nhau.

Cấu trúc dữ liệu va chạm cần cung cấp cho thư viện được định nghĩa như phía dưới. Ngoài ra bạn còn có thể định nghĩa miễn là thỏa mãn các yêu cầu sau :

10.1 Điểm tiếp xúc

Nếu hai vật va chạm, hoặc một vật va chạm với giá, vùng tiếp xúc được biểu diễn bởi một hoặc nhiều điểm tiepé xúc ”contact points”. Điểm tiếp xúc được định nghĩa theo cấu trúc dContactGeom:

```
struct dContactGeom {
      dVector3 pos; // V• trí ti•p xúc
      dVector3 normal; // Vector pháp
      dReal depth; // •• sâu va ch•m
      dGeomID g1,g2; //  Hình d•ng hình h•c c•a v•t
};
```

`pos` lưu vị trí tiếp xúc trong hệ tọa độ tuyệt đối.

depth độ sau mà hai vật thâm nhập vào nhau. Nếu `depth`=0 thì hai vật chỉ chạm nhau. Tuy nhiên đây là trường hợp hiếm có trong mô phỏng mà theo thời gian các vật sẽ thâm nhập vao nhau làm cho depth khác 0.

`normal` là vector đơn vị, là vector chỉ phương của bề mặt tiếp xúc.

`g1` và `g2` là hình học va chạm nhau.

Theo quy ước là nếu body1 dịch chuyển theo vector pháp tuyến một khoảng bằng chiều sâu depth (hoặc tương đương nếu body2 dịch chuyển cùng một lượng với body1 nhưng theo chiều ngược lại) thì độ sâu tiếp xúc sẽ giảm dần về 0. Nghĩa là vector pháp tuyến có gốc đặt tại body1.

Trong thực tế, tiếp xúc giữa hai vật là một điều phức tạp. Việc lấy đại diện các điểm tiếp xúc đó chỉ là xấp xỉ. Việc lấy dữ liệu tiếp xúc là một miếng hay một mặt là chính xác về mặt vật lý hơn, nhưng điều này vẫn còn là một thách thức trong mô phỏng tốc độ cao.

Khi một điểm tiếp xúc được thêm vào không gian mô phỏng làm cho tốc độ mô phỏng giảm đi, cho nên nhiều khi ta phải bỏ qua các điểm tiếp xúc để có được tốc độ mô phỏng. Ví dụ, khi hai hộp chữ nhật va chạm sẽ có nhiều điểm tiếp xúc cần được miêu tả đúng trạng thái hình học khi va chạm, nhưng ta chọn 3 điêm rtốt nhất. Nghĩa là chúng ta sẽ lấy những giá trị tốt nhất trong tập điểm xấp xỉ.

10.2 Hình dạng hình học (Geom)

Hình học của vật thể là dữ liệu nền tảng trong không gian hình học có tính toán va chạm. Hình học có thể miêu tả hình dạng của vật thể đơn (như hình hộp, hình cầu), hoặc để miêu tả một nhóm vật thể phúc tạp. Một số hình có thể dịch chuyển, một số thì không. Các hình động dịch chuyển có vector định vị và một ma trận quay 3*3, chúng có thể thay đổi vị trí trong khi mô phỏng, còn hình tĩnh thì không nó đại diện cho các giá và không di chuyển được, Với mỗi hình đều có vị trí và hướng nhưng chúng không thể tự định vị trong không gian mô phỏng, bởi vật chúng chỉ đại diện cho vật về mặt hình học, nên khi tính toán chúng lấy vị trí và hướng từ vật thể.

Chú ý geoms khác với body chúng chỉ có các thuộc tính hình học (kích thước, hình dạnh, vị trí và hướng) nhưng không có các thuộc tính vật lý (như vận tốc và khối lượng). Cả body và geom đi cùng với nhau để miêu tả đủ thuộc tính của vật cần mô phỏng.

10.3 Không gian hình học (Space)

Không gian hình học "Space" nơi chứa các loại hình vật khác nhau, trong không gian này việc tìm kiếm va chạm giữa các hình rất nhanh. Nếu ở ngoài không gian để tìm va chạm trong khi mô phỏng phải gọi hàm `dCollide()` để nhận được điểm tiếp xúc ở mỗi cặp va chạm. Với N hình ta phải gọi O(N2) lần kiểm tra, điều này rất vất vả nếu cơ hệ có nhiều vật. Một cách tiếp cận tốt hơn là chèn hình vào không gian bằng cách gọi hàm `dSpaceCollide()`. Không gian sẽ thực hiện tìm kiếm va chạm có chọn lọc, nghĩa là nó sẽ nhanh chóng xác định được các cặp hình học có nguy cơ va chạm nhau. Các cặp đó sẽ được bỏ qua khi gọi lại các hàm, nó sẽ gọi hàm `dCollide()` cho những cặp đó. Nó tiết kiệm nhiều thời gian hơn khi cứ phải gọi hàm `dCollide()` để kiểm tra, ở số lượng các cặp nguy cơ va chạm nhỏ hơn nhiều số cặp trong không gian. Một Spaces có thể chứa nhiều spaces con. Điều này rất hữu ích cho việc chia nhỏ không gian va chạm theo dạng phân cấp để xúc tiến việc tối ưu và tăng tốc độ tìm va chạm.

10.4 Hàm về hình dạng hình học

Các hàm sau có tác dụng đối với tất cả các mô hình hình học.

```
void dGeomDestroy (dGeomID);
```

Xóa bỏ mô hình hình học, trước hết hủy bỏ ra khỏi không gian “space”. Hàm này xóa bỏ mọi kiểu hình học, nhưng để tạo mô hình hình học bạn phải gọi hàm khởi tạo cho từng kiểu mô hình. Khi không gian “space” bị xóa bỏ, các hình học trong nó cũng tự động bị xóa bỏ.

```
void dGeomSetData (dGeomID, void *);
void *dGeomGetData (dGeomID);
```

Hai hàm gán và nhận cho trỏ dữ liệu người dùng định nghĩa mô hình hình học.

```
void dGeomSetBody (dGeomID, dBodyID);
dBodyID dGeomGetBody (dGeomID);
```

Hai hàm gán và nhận vật được gắn với mô hình hình học, khi thực hiện hàm gán, sẽ tự động tính toán để sao cho vị trí và hướng của vật và mô hình hình học là như nhau. Nếu tham biến dbodyID là 0 ta được vị trí và hướng của mô hình độc lập với vật. Nếu mô hình đã được thiết lập với vật thì khi gắn với vật khác thì vị trí và hướng sẽ tương ứng với vật này.

```
void dGeomSetPosition (dGeomID, dReal x, dReal y, dReal z);
void dGeomSetRotation (dGeomID, const dMatrix3 R);
void dGeomSetQuaternion (dGeomID, const dQuaternion);
```

Thiết lập vị trí,, ma trận quay hoặc quaternion cho mô hình hình học. Hàm này trương tự như các hàm `dBodySetPosition()`, `dBodySetRotation()` và `dBodySetQuaternion()`. Nếu “geom” được nối với body, thì vị trí và ma trận quay, quaternion cũng sẽ thay đổi.

```
const dReal * dGeomGetPosition (dGeomID);
const dReal * dGeomGetRotation (dGeomID);
void dGeomGetQuaternion (dGeomID, dQuaternion result);
```

Hai hàm đầu trả về con trỏ vị trí và ma trận quay của “geom”. Những giá trị được trả lại là những con trỏ tới những cấu trúc dữ liệu bên trong, nếu “geom” nối với body khi body các giá trị vị trí và ma trận quay của body thay đổi thì con trỏ này cũng thay đổi theo lúc này ta phải gọi `dBodyGetPosition()` or `dBodyGetRotation()`. `dGeomGetQuaternion()` để cập nhật các giá trị này vào trong “space”. Nếu gọi các hàm này cho hình học cố định (Miêu tả mô hình hình học của gia) sẽ trả về lỗi trong chế độ debug

```
void dGeomGetAABB (dGeomID, dReal aabb[6]);
```

Hàm trả về aabb là đinghr của hình học bao quanh mô hình. Mảng aabb là các phần tử (minx,maxx, miny, maxy, minz, maxz).

```
int dGeomIsSpace (dGeomID);
```

Trả về 1 nếu “geom” ở trong không gian “space”, hoặc 0 nếu không thuộc.

```
dSpaceID dGeomGetSpace (dGeomID);
```

Trả lại “space” chứa mô hình, hoặc trả về 0 nếu mô hình không thuộc bất kỳ không gian nào.

```
int dGeomGetClass (dGeomID);
```

Trả về số hiệu nhận dạng đối tượng, Các đối tượng cơ bản là:

dSphereClass	Hình cầu
dBoxClass	Hình hộp
dCCylinderClass	Hình trụ có mũ bán cầu
dCylinderClass	Hình trụ
dPlaneClass	Mặt phẳng (non-placeable)
dGeomTransformClass	Biến đổi hình học
dRayClass	Tia thẳng
dTriMeshClass	Mặt tam giác
dSimpleSpaceClass	Không gian đơn giản
dHashSpaceClass	Không gian hỗn độn

```
void dGeomSetCategoryBits (dGeomID, unsigned long bits);
void dGeomSetCollideBits (dGeomID, unsigned long bits);
unsigned long dGeomGetCategoryBits (dGeomID);
unsigned long dGeomGetCollideBits (dGeomID);
```

Các đối tương tự định nghĩa sẽ trả về các giá trị do người dùng quy định. Theo các hàm sau.

```
void dGeomEnable (dGeomID);
void dGeomDisable (dGeomID);
int dGeomIsEnabled (dGeomID);
```

Cho phép tính toán hoặc bỏ qua mô hình khi tính toán va chạm, bởi gọi các hàm dSpaceCollide() và dSpaceCollide2(), mặc dù chúng vẫn là các đối tượng thuộc không gian mô hình “space”.

dGeomIsEnabled() trả về 1 nếu "geom" là enabled hoặc không là disabled. Khi "geoms" được tạo chúng mạc định là được tính toán va chạm.

10.5 Kiểm tra va chạm

Để kiểm tra va chạm cho cơ hệ ta phải tạo ra một không gian rồi đưa các mô hình hình học vào trong nó. Tại mỗi bước mô phỏng ta muốn tính toán muốn tìm tất cả các mặt tiếp xúc của các mô hình va chạm nhau. Có ba hàm có thể được sử dụng như sau:

dCollide() kiểm tra hai mô hình cắt nhau và tính toán các điểm tiếp xúc.

dSpaceCollide() xác định các cặp mô hình trong không gian có nguy cơ va chạm nhau, và gọi hàm callback cho mỗi cặp đó. Nó sẽ không tính toán trực tiếp các điểm va chạm, bởi người dủng chỉ muốn xác định một vài cặp va chạm đặc biệt – ví dụ họ có thể lờ đi hoặc có những quyết định khác. Những quyết như vậy được làm trong hàm callback, nó có thể cân nhắc cần hoặc không cần phải gọi hàm dCollide().

dSpaceCollide2() xác định xem các mô hình khôgn không gian này có nguy cơ va chạm với các mô hình trong không gian khác không, và gọi các hàm callback cho mỗi cặp có nguy cơ va chạm. Chúng có thể kiểm tra các mô hình đơn với một không gian mô hình. Hàm này rất hữu ích khi có các van chạm phân cấp, nơi mà một không gian chứa các không gian con.

Hệ thống va chạm đã được thiết kế để đa cho người dùng tính linh hoạt và có thể quyết định rằng những đối tượng nào sẽ được kiểm tra va chạm lẫn nhau. Tại sao ta lại viết riêng ba hàm va chạm thay vì một hàm kiểm tra tất cả các cặp tiếp xúc. Không gian hình học có thể chứa đựng những không gian khác. Những không gian đó đại diện tập hợp tiêu biểu mô hình hình học. Được đặt cạnh nhau khi mô phỏng ta phải kiểm tra các khả năng va chạm của tất cả các cặp mô hình nếu chỉ dùng hàm dCollide() thi việc lưu trữ các điểm tiếp xúc là rất lớn. Ví dụ bạn cần điều khiển hai ô tô trên một địa hình nào đó, mỗi mô hình ô tô được tạo ra bởi nhiều các mặt hình học và nếu cùng được chèn vào cùng một không gian thì tính toán va chạm giữa hai ô tô bằng tổng các va chạm của các mặt tạo ra hai mô hình ô tô (có khi còn bằng bình phương của số này, phụ thuộc vào kiểu không chứa mô hình).

Để tăng tốc độ tính toán va chạm ta tạo ra những không gian riêng biệt đại diện cho mỗi ô to. Các mặt hình học của mô hình ô to được chèn vào không gian tương ứng đó, và không gian này được chèn vào không gian chung nào đó. Tại mỗi bước mô phỏng hàm dSpaceCollide() được gọi cho không gian chung này. Điều này tạo ra việc kiểm tra giữa cặp không gian của hai ô tô (giữa hai hộp bao quanh ô tô) và gọi hàm callback nếu chúng va chạm. Hàm này sẽ kiểm tra các mặt hình học trong không gian ô tô xem có va chạm nhau không bằng cách gọi hàm dSpaceCollide2(). Nếu hai ô tô không gần nhau

Dưới đây một hàm callback mẫu xuyên qua tất cả các không gian và những không gian con, tính toán tất cả những điểm tiếp xúc cho mọi mặt hình học cắt nhau:

```
void nearCallback (void *data, dGeomID o1, dGeomID o2)
{
      if (dGeomIsSpace (o1) || dGeomIsSpace (o2))
      {
            // Tính va ch•m c•a không gian có các m•t hình h•c
            dSpaceCollide2 (o1,o2,data,&nearCallback);
            // Tính va ch•m các m•t hình h•c trong n•i t•i không gian
            if (dGeomIsSpace (o1)) dSpaceCollide (o1,data,&nearCallback);
            if (dGeomIsSpace (o2)) dSpaceCollide (o2,data,&nearCallback);
      }
      else {
            // Tính va ch•m c•a hai v•t không thu•c không gian, tìm •i•m va ch•m
            // •i•m gi•a o1 và o2
            int num_contact = dCollide (o1,o2,max_contacts,contact_array,skip);
            // ••a •i•m tiếp xúc vào không gian mô ph•ng
            ...
      }
}
```

...

// Va chạm giữa các cặp hình học

```
dSpaceCollide (top_level_space,0,&nearCallback);
```

Hàm callback của "space" không cho phép hiệu chỉnh không gian "space" trong khi làm việc với hàm dSpaceCollide() hoặc dSpaceCollide2(). Ví dụ, bạn không thể thêm hoặc bớt đối tượng hình học từ không gian này và khôgn thể thay đổi vị trí đối tượng. Cố tình sẽ gay lỗi trong chế độ biên dịch debug.

10.5.1 Category và Collide bitfields

Mỗi mô hình hình học đều có "category" và "collide" bitfield được dùng hỗ trợ các thuật toán không gian để xác định "geoms" cắt nhau hoặc không. Thuộc tính này được mạc định đi theo "geom" xác định tính có khả năng va chạm với cả các "geom" khác.

Mỗi cặp mô hình hình học geoms sẽ được xem xét bởi hai hàm dSpaceCollide() và dSpaceCollide2() để chuyển cho hàm callback chỉ khi chúng đụng nhau. Khi lapạ trình ta có thể tham khả các đoạn mã sau:

// Kiểm tra nếu mô hình o1 và mô hình o2 có thể va chạm nhau

```
cat1 = dGeomGetCategoryBits (o1);
cat2 = dGeomGetCategoryBits (o2);
col1 = dGeomGetCollideBits (o1);
col2 = dGeomGetCollideBits (o2);
if ((cat1 & col2) || (cat2 & col1))
{
      // g•i các hàm callback cho o1 và  o2
}
else {
      // Không làm gì c• n•u  o1và  o2 khôgn va ch•m nhau
}
```

Lưu ý rằng chỉ hai hàm dSpaceCollide() và dSpaceCollide2() dùng khái niệm bitfields, không dùng cho hàm dCollide().

10.5.2 Hàm kiểm tra va chạm

```
int dCollide (dGeomID o1, dGeomID o2, int flags,
              dContactGeom *contact, int skip);
```

Hàm này lấy hai mô hình `o1` và `o2` kiểm tra chúng xem có khả năng va chạm nhau không, sinh ra thông tin va chạm. Tham số `flags` xác định số lượng điểm tiếp xúc khi hai mô hình va chạm nhau. Chú ý rằng nếu số này là 0, hàm này không cho thông tin về sự va chạm. Các điểm tiếp xúc được lưu vào mảng có cấu trúc `dContactGeom`, tham số skip có giá trị bằng sizeof(`dContactGeom`) nếu khác

Nếu o1 và o2 là cùng một mô hình thì hàm này sẽ không làm gì và trả lại 0. Về mặt kỹ thuật việc một đối tượngv tự cắt với chính nó là không thể xẩy ra. Hàm này không quan tâm o1 và o2 có cùng trong không gian hay không.

```
void dSpaceCollide (dSpaceID space,
                    void *data, dNearCallback *callback);
```

Hàm này kiểm tra hai đối tượng hình học có nguy cơ va chạm nhau không, và gọi lại nếu chúng va chạm. Hàm callback có kiểu dNearCallback, và được định nghĩa như sau:

```
dNearCallback (void *data, dGeomID o1, dGeomID o2);
```

con trỏ *data sẽ được duyệt bởi hàm dSpaceCollide(). Nghĩa là bạn có thể định nghĩa con trỏ này. Còn hai tham số o1 và o2 là hai mô hình có khoảng cách gần nhau có nguy cơ va chạm.
Hàm callback có thể gọi hàm dCollide() để kiểm tra o1 và o2 và sinh ra tập điểm va chạm (contact points) nếu chúng va chạm nhau. Sau đó tập điểm này có thể được thêm vào để mô phỏng qua trình vật lý xẩy ra trong không gian của cơ hệ. Người dùng có thể không gọi cho gọi hàm dCollide() đối với bất kỳ cặp mô hình nào mà họ cho rằng không quan trọng.

```
void dSpaceCollide2 (dGeomID o1, dGeomID o2,
                     void *data, dNearCallback *callback);
```

Hàm này khá giống hàm dSpaceCollide(), chỉ có điều nó lấy hai geoms (hoặc không gian hình học) là tham số để kiểm tra. Nó gọi callback cho tất cả các cặp giao nhau tiềm tàng mà chứa đựng một geom từ o1 và geom từ o2.

Độ chính xác phụ thuộc vào những kiểu o1 Và o2:

• Nếu tham số là geom và space khác, hàm callback kiểm tra tất cả các khả năng sự giao nhau giữa những geom và đối tượng trong space.

• Nếu cả o1 lẫn o2 là space, hàm callback kiểm tra tất cả các cặp giao nhau tiềm tàng giữa mỗi geom từ o1 và mỗi geom từ o2. Giải thuật mà được sử dụng phụ thuộc vào loại space. Nếu không có giải thuật nào tối ưu thì hàm này sẽ dùng đến một trong số cách sau :

1. Tất cả geoms trong o1 được kiểm tra với từng geom trong o2.

2. Tất cả geoms trong o2 được kiểm tra với từng geom trong o1.

Cách được dùng phụ thuộc vào một số quy tắcsau

• Nếu cả tham số là space giống nhau, ta sẽ gọi hàm dSpaceCollide().

• Nếu cả hai tham số không phải là space, ta gọi hàm callback một lần với những tham số này.

Nếu hàm này có hai tham số là space và geom X cùng trong một không gian, trường hợp này callback sẽ luôn luôn được gọi với cặp (X,X), bởi vì một đối tượng luôn luôn cắt chính nó. Người dùng có thể hoặc lờ đi không kiểm tra, hoặc chỉ cần đưa tham số (X,X) Tới hàm `dCollide()` hàm này sẽ trả lại NULL).

10.6 Hàm về không gian hình học

Có vài loại không gian (space). Mỗi loại sử dụng những cấu trúc dữ liệu bên trong khác nhau để cất giữ geoms, Và những giải thuật khác nhau cho việc thực hiện kiểm tra sự va chạm có chọn lọc :

• Không gian đơn giản (simple space). Không gian này kiểm tra tất các khả năng va chạm - nó đơn giản kiểm tra mỗi cặp geoms có thể va chạm nhau và báo lại những cặp có AABBs gối lên nhau. Thời gian yêu cầu kiểm tra va chạm của n đối tượng là O(n2). Không gian kiểu này không dùng được khí có nhiều đối tượng, nhưng nó có thể là giải thuật thích hợp cho một số lượng nhỏ đối tượng. Ngoài ra để gỡ lỗi tiềm tàng của hệ thống va chạm.

• Không gian hỗn độn (hash space). Không gian này sử dụng một cấu trúc dữ liệu bên trong mà lưu mỗi geom chồng lên những ô lưới 3 chiều. Mỗi lưới như một hình lập phương có cạnh lengths bằng 2^i, Trong đó i là một số nguyên mà hạn chế từ một tối thiểu đến một trị số cực đại. Thời gian yêu cầu kiểm tra giao nhau cho n đối tượng là O(n).

• Không gian Quadtree. Không gian này sử dụng dữ liệu hình cây với mỗi nút là một ô lưới AABB để nhanh chóng lọc đối tượng để kiểm tra va chạm. Phù hợp với việc mô phỏng chuyển động của đối tượng trên nền địa hình. Hiện thời hàm `dSpaceGetGeom()` chưa thực hiện cho không gian quadtree.

Dưới đây những hàm được sử dụng cho những không gian :

```
dSpaceID dSimpleSpaceCreate (dSpaceID space);

dSpaceID dHashSpaceCreate (dSpaceID space);
```

Tạo ra một không gian, Simple space hoặc Hash space. Nếu `space` khác không, sẽ chèn không gian mới vào trong không gian đó.

```
dSpaceID dQuadTreeSpaceCreate (dSpaceID space, dVector3 Center,
                               dVector3 Extents, int Depth);
```

Tạo ra một không gian quadtree. gồm tâm `center` và phạm vi `(Extents)` định nghĩa kích thước nút gốc. `Depth` đặt chiều sâu cây - số lượng những khối mà đọc tạo ra là `4 ^Depth`.

```
void dSpaceDestroy (dSpaceID);
```

Hàm hủy một space. Hàm này giống với hàm `dGeomDestroy()` chỉ có điều tham số là SpaceID. Khi một space được hủy bỏ, nếu kiểu cleanup của nó là 1 (Mặc định). Thì tất cả geoms trong space tự động được hủy bỏ theo.

```
void dHashSpaceSetLevels (dSpaceID space, int minlevel, int maxlevel);
void dHashSpaceGetLevels (dSpaceID space, int *minlevel, int *maxlevel);
```

Hàm set và get các tham số kích thước lớn nhất và nhỏ nhất ô lưới trong không gian `HashSpace` sẽ là `2 ^. minlevel` và `2 ^. maxlevel` vào luôn thảo mãn `minlevel` phải nhỏ hơn hoặc bằng `maxlevel`.

```
void dSpaceSetCleanup (dSpaceID space, int mode);
int dSpaceGetCleanup (dSpaceID space);
```

Hàm set và get kiểu cleanup của space. Nếu kiểu cleanup là 1, thì geoms được tự động phá hủy khi space được phá hủy. Nếu kiểu cleanup là 0 điều này sẽ không xẩy ra. Kiểu cleanup mặc định cho nhữngspace mới là 1.

```
void dSpaceAdd (dSpaceID, dGeomID);
```

Hàm cho thêm một geom vào một space. Hàm này sẽ không thực hiện nếu geom đã nằm trong space . Hàm này có thể được gọi tự động nếu tham số là hàm khởi tạo geom.

```
void dSpaceRemove (dSpaceID, dGeomID);
```

Bỏ một geom ra khỏi space. Hàm này sẽ không thực hiện nếu geom không nằm trong space . Hàm này có thể được gọi tự động nếu tham số là hàm hủy dGeomDestroy()..

```
int dSpaceQuery (dSpaceID, dGeomID);
```

Hàm này trả lại 1 nếu geom đã cho nằm trong space, hoặc trả lại 0 nếu nó không.

```
int dSpaceGetNumGeoms (dSpaceID);
```

Trả lại số lượng geoms trong một không gian.

```
dGeomID dSpaceGetGeom (dSpaceID, int i);
```

Trả lại chỉ số của geom nằm trong space. i phải hạn chế từ 0 đến `dSpaceGetNumGeoms().` Nếu bất kỳ sự thay đổi nào làm cho số lượng geom trong space thay đổi (bao gồm thêm và xóa geoms) thì hàm không bảo đảm có thể luôn trả về đúng geom trong không gian.

Hàm này thực hiện nhanh khi geoms được truy nhập theo thứ tự 0,1,2,... Những thứ tự không tuần tự khác có thể kết quả truy nhập chậm hơn, phụ thuộc vào sự thi hành bên trong.

10.7 Lớp đối tượng hình học

10.7.1 Sphere class

```
dGeomID dCreateSphere (dSpaceID space, dReal radius);
```

Tạo ra một geom hình cầu có bán kính `radius`, và trả lại chỉ số của nó. Nếu space khác không, chèn nó vào trong space đó. Điểm tham khảo của hình cầu là tâm nó.

```
void dGeomSphereSetRadius (dGeomID sphere, dReal radius);
```

Đặt bán kính hình cầu cho chỉ số sphere.

```
dReal dGeomSphereGetRadius (dGeomID sphere);
```

Trả lại bán kính của hình cầu.

```
dReal dGeomSpherePointDepth (dGeomID sphere, dReal x, dReal y, dReal z);
```

Trả lại chiều sâu khoảng cách điểm (x,y,z) so với mặt cầu. Những điểm ở trong geom sẽ có chiều sâu dương, bên ngoài nó sẽ có chiều sâu âm, và những điểm trên bề mặt sẽ có chiều sâu là 0.

10.7.2 Box class

```
dGeomID dCreateBox (dSpaceID space, dReal lx, dReal ly, dReal lz);
```

Tạo ra một geom hình hộp chữ nhật có độ dài các cạnh `lengths ( lx,ly,lz),` và trả lại chỉ số của nó. Nếu space khác không, chèn nó vào trong space đó. Điểm tham khảo của hình hộp là trọng tâm nó.

```
void dGeomBoxSetLengths (dGeomID box, dReal lx, dReal ly, dReal lz);
```

Hàm set độ dài các cạnh Lengths của hình hộp.

```
void dGeomBoxGetLengths (dGeomID box, dVector3 result);
```

Hàm trả lại độ dài các cạnh lengths hình hộp.

```
dReal dGeomBoxPointDepth (dGeomID box, dReal x, dReal y, dReal z);
```

Trả lại chiều sâu khoảng cách điểm `(x,y,z)` so với mặt hình hộp. Những điểm ở trong geom sẽ có chiều sâu dương, bên ngoài nó sẽ có chiều sâu âm, và những điểm trên bề mặt sẽ có chiều sâu là 0.

10.7.3 Plane class

```
dGeomID dCreatePlane (dSpaceID space,
                      dReal a, dReal b, dReal c, dReal d);
```

Tạo ra một geom mặt phẳng với những tham số đã cho, và trả lại chỉ số của nó. Nếu space khác không, chèn nó vào trong không gian đó.
Phong trình mặt phẳng: $a x + b y + c z = d$

Vector chỉ phương của mặt phẳng là (a, b,c), và phải có độ dài bằng 1. Các tham số (a,b,c,d) luôn luôn trong tọa độ toàn cầu. Trong những trường hợp thực tế mặt phẳng luôn luôn là giá và không bị ràng buộc tới bất kỳ đối tượng chuyển động được nào.

```
void dGeomPlaneSetParams (dGeomID plane, dReal a, dReal b, dReal c, dReal d);
```

Hàm Set tham số tạo mặt phẳng.

```
void dGeomPlaneGetParams (dGeomID plane, dVector4 result);
```

Hàm trả lại các tham số khởi tạo mặt phẳng

```
dReal dGeomPlanePointDepth (dGeomID plane, dReal x, dReal y, dReal z);
```

Trả lại chiều sâu khoảng cách điểm (x,y,z) so với mặt phẳng. Những điểm cùng chiều vector chỉ phương geom sẽ có chiều sâu dương, bên ngoài nó sẽ có chiều sâu âm, và những điểm trên bề mặt sẽ có chiều sâu là 0.

10.7.4 Capped cylinder class

```
dGeomID dCreateCCylinder (dSpaceID space, dReal radius, dReal length);
```

Tạo ra một geom hình trụ đội mũ bán cầu với những tham số đã cho, và trả lại chỉ số của nó. Nếu `space` khác không, chèn nó vào trong không gian đó.

Một hình trụ có mũ bán cầu như một hình trụ bình thường trừ nó có thêm một nửa cầu ở hai đầu. Đặc tính này làm cho việc dò tìm va chạm nhanh và chính xác.

```
void dGeomCCylinderSetParams (dGeomID ccylinder,
                              dReal radius, dReal length);
```

Hàm đặt tham số khởi tạo gồm có bán kính và chiều dài hình trụ.

```
void dGeomCCylinderGetParams (dGeomID ccylinder,
                              dReal *radius, dReal *length);
```

Hàm lấy tham số gồm có bán kính là `radius` và chiều dài `length` hình trụ.

```
dReal dGeomCCylinderPointDepth (dGeomID ccylinder,
                                dReal x, dReal y, dReal z);
```

Trả lại chiều sâu khoảng cách điểm `(x,y,z)` so với mặt trụ. Những điểm cùng nằm trong geom sẽ có chiều sâu dương, bên ngoài nó sẽ có chiều sâu âm, và những điểm trên bề mặt sẽ có chiều sâu là 0.

10.7.5 Ray class

Một đối tượng Ray khác với tất cả các lớp geom khác nhau. Đó là một nửa đường thẳng vô tận mà gốc xuất phát là điểm khởi tạo geom và đầu kia kéo dài vô tận.

Khi gọi hàm `dCollide()` để kiểm tra va chạm giữa Ray và Geom sẽ cho kết quả khác với đa số các điểm tiếp xúc của các đối tượng khác. Những Ray có những quy ước riêng về thông tin tiếp xúc trong cấu trúc dContactGeom:

• pos - điểm mà Ray cắt ngang bề mặt geom, bất chấp Ray bắt đầu từ ở trong hoặc bên ngoài geom.

• Normal - đây là vector chỉ phương của geom tại điểm tiếp xúc.
• Depth- đây là khoảng cách từ điểm gốc của Ray đến điểm tiếp xúc.

Đối tượng Ray rất hữu ích cho các thứ định hướng những vật chuyển động hoặc cho sự xếp đặt đối tượng.

```
dGeomID dCreateRay (dSpaceID space, dReal length);
```

Tạo ra một geom kiểu Ray những tham số length đã cho, và trả lại chỉ số của nó. Nếu space khác không, chèn nó vào trong không gian đó.

```
void dGeomRaySetLength (dGeomID ray, dReal length);
```
Hàm đặt tham số length để khởi tạo `ray`.

```
dReal dGeomRayGetLength (dGeomID ray);
```
Hàm lấy tham số length của ray.

```
void dGeomRaySet (dGeomID ray, dReal px, dReal py, dReal pz,
                  dReal dx, dReal dy, dReal dz);
```

Đặt vị trí bắt đầu `(px,py,pz)` và phương hướng `(dx,dy,dz)` cho `Ray`. Ma trận quay của `Ray` sẽ được điều chỉnh để trục Z trùng với phương hướng của `Ray`. Chú ý rằng hàm này không điều chỉnh `length` của `ray`.

```
void dGeomRayGet (dGeomID ray, dVector3 start, dVector3 dir);
```

Hàm trả về vị trí đầu `(start)` và phương hướng `(dir)` (của) tia `Ray`. Vector `dir` được trả lại sẽ là một vectơ đơn vị.

10.7.6 Triangle Mesh class

Một lưới tam giác (TriMesh) là tập hợp những hình tam giác. Khi mô tả đối tượng hình học theo lưới tam giác thì hệ thống có những đặc tính sau:

- Một mặt tam giác bất kỳ khi được sử dụng không yêu cầu phải theo cấu trúc strip, fan hoặc grid.
- Những mắt lới tam giác có thể tương tác với những hình cầu, hình hộp, Ray và lưới tam giác khác.
- Làm việc tốt khi mô tả lưới tam giác có kích thước khá lớn.

• Nó sử dụng liên kết hành tinh để tăng tốc độ kiểm tra va chạm. Khi một geom được kiểm tra với một trimesh khác, dữ liệu được cất giữ ngay bên trong trimesh. Dữ liệu này có thể được xóa khi dùng hàm dGeomTriMeshClearTCCache().

Kiểu va chạm Trimesh / Trimesh, thực hiện khá tốt, nhưng có ba cảnh báo sau :

• Bước tính stepsize khi sử dụng, nói chung, phải đọc giảm bớt khi tính toán va chạm yêu cầu chính xác. Sự va chạm hình bao không lồi (non-convex) phụ thuộc nhiều hơn vào bước tính (stepsize) so với hình bao lồi hoặc các đối tượng cơ bản. Hơn nữa, các tiếp xúc cục bộ cũng thay đổi nhanh chóng hơn đối với những hình cầu và những lập phương.

• Để giải quyết hiệu quả những va chạm, hàm dCollideTTL cần những vị trí đụng nhau của các Trimesh trong timestep trước (bước tính trước). Mục đích là dùng để tính toán va chạm của các Trimesh chuyển động có vận tốc, yêu cầu cần phải có phương va chạm và vector chỉ phương tại điểm tiếp xúc,... Các thông số này yêu cầu người lập trình cập nhật vào trong những biến này tại mỗi bước tính timestep. Sự cập nhật này được thực hiện bên ngoài thư viện, vì vậy nó không được đưa vào trong thư viện. Đoạn mã cập nhật này có thể như sau:

```
const double *DoubleArrayPtr =Bodies[BodyIndex].TransformationMatrix->GetArray();
dGeomTriMeshDataSet( TriMeshData,
                     TRIMESH_LAST_TRANSFORMATION,
                     (void *) DoubleArrayPtr );
```

Ma trận biến đổi có chuẩn là 4x4, Và ” DoubleArray ” Là mảng một chiều lưu 16 giá trị ma trận.

Chú ý: Các hàm về trimesh còn được sửa đổi tiếp trong các phiên bản tiếp theo cho hoàn chỉnh.

```
dTriMeshDataID dGeomTriMeshDataCreate();
void dGeomTriMeshDataDestroy (dTriMeshDataID g);
```

Hàm khởi tạo và xóa bỏ Trimesh các tham số về Trimesh được lưu giữ theo kiểu `dTriMeshDataID`.

```
void dGeomTriMeshDataBuild (dTriMeshDataID g, const void* Vertices,
                            int VertexStride, int VertexCount,
                            const void* Indices, int IndexCount,
                            int TriStride, const void* Normals);
```

Điền dữ liệu vào đối tượng dTriMeshData. Không có dữ liệu nào được sao chép ở đây, Ở đây con trỏ chỉ vào địa chỉ vùng dữ liệu tạo lưới triangle. Các câu lệnh sau miêu tả cách thức khởi tạo dữ liệu:

```
struct StridedVertex {
      dVector3 Vertex; // 4th component can be left out, reducing memory usage
      // Userdata
```

```
};
int VertexStride = sizeof (StridedVertex);
struct StridedTri {
      int Indices[3];
      // Userdata
};
int TriStride = sizeof (StridedTri);
```

Tham số Normal là vector chỉ phượng của các đối tượng trimesh. Ví dụ,

```
dTriMeshDataID TriMeshData;
  TriMeshData = dGeomTriMeshGetTriMeshDataID (
        Bodies[BodyIndex].GeomID);

  // as long as dReal == floats
  dGeomTriMeshDataBuildSingle (TriMeshData,
        // Vertices
        Bodies[BodyIndex].VertexPositions,
        3*sizeof(dReal), (int) numVertices,
        // Faces
        Bodies[BodyIndex].TriangleIndices,
        (int) NumTriangles, 3*sizeof(unsigned int),
        // Normals
        Bodies[BodyIndex].FaceNormals);
```

Các giá trị này sẽ được tính toán trước khi tính toán các vị trí tiếp xúc giữa các vật, nhưng là không cần thiết nếu bạn không muốn tính vector normals của các mặt triangle trước khi xây dựng dữ liệu. Nghĩa là tham số này có thể đặt bằng NULL, hàm dCollideTTL sẽ chú ý chính toán vector này.

```
void dGeomTriMeshDataBuildSimple (dTriMeshDataID g, const dVector3*Vertices,
                                  int VertexCount, const int* Indices,
                                  int IndexCount);
```

Hàm xây dựng dữ liệu `dTriMeshDataID g` chuẩn bị khởi tạoTrimesh.

```
typedef int dTriCallback (dGeomID TriMesh, dGeomID RefObject, int TriangleIndex);
void dGeomTriMeshSetCallback (dGeomID g, dTriCallback *Callback);
```

```
dTriCallback* dGeomTriMeshGetCallback (dGeomID g);
```

Cho phép chỉ định tam giác khi cần callback. Cho phép người dùng kiểm soát va chạm với một hình tam giác đặc biệt nào đó. Nếu giá trị trở lại là zero không có sự tiếp xúc nào sẽ được phát sinh.

```
typedef void dTriArrayCallback (dGeomID TriMesh, dGeomID RefObject,
                                const int* TriIndices, int TriCount);
void dGeomTriMeshSetArrayCallback (dGeomID g, dTriArrayCallback* ArrayCallback);
dTriArrayCallback *dGeomTriMeshGetArrayCallback (dGeomID g);
```

Để chọn Geom callback. Cho phép người dùng làm chủ danh sách tất cả các hình tam giác giao nhau.

```
typedef int dTriRayCallback (dGeomID TriMesh, dGeomID Ray, int TriangleIndex,
                             dReal u, dReal v);
void dGeomTriMeshSetRayCallback (dGeomID g, dTriRayCallback* Callback);
dTriRayCallback *dGeomTriMeshGetRayCallback (dGeomID g);
```

Hàm cho phép kiểm tra đối tượng Ray có va chạm với Trimesh không.

```
dGeomID dCreateTriMesh (dSpaceID space, dTriMeshDataID Data,
                        dTriCallback *Callback,
                        dTriArrayCallback * ArrayCallback,
                        dTriRayCallback* RayCallback);
```

Khởi tạo đối tượng TriMesh sẽ sử dụng để tính toán va chạm.

```
void dGeomTriMeshSetData (dGeomID g, dTriMeshDataID Data);
```

Thay thế dữ liệu hiện tại của TriaMesh.

```
void dGeomTriMeshClearTCCache (dGeomID g);
```

Làm sạch bộ đệm Cache của TriMesh g.

```
void dGeomTriMeshGetTriangle (dGeomID g, int Index, dVector3 *v0,
                              dVector3 *v1, dVector3 *v2);
```

Hàm đỉnh của tam giác thứ `index` của đối tượng TriMesh dữ liệu được lưu vào `v0, v1, v2`.

```
void dGeomTriMeshGetPoint (dGeomID g, int Index, dReal u, dReal v,
                           dVector3 Out);
```

Hàm lấy vị trí tam giác thứ `index` của đối tượng TriMesh dữ liệu được lưu vào `Out`.

```
void dGeomTriMeshEnableTC(dGeomID g, int geomClass, int enable);
int dGeomTriMeshIsTCEnabled(dGeomID g, int geomClass);
```

Những hàm này có thể cho phép / Vô hiệu hóa việc sử dụng các quan hệ hành tinh giữa các Triangle với nhau khi kiểm tra va chạm. Quan hệ hành tinh này có thể cho phép /Vô hiệu khi kiểm tra giữa các cặp Triangle / geom, hiện thời nó làm việc với hình cầu và hình hộp. Mặc định cho những hình cầu.

Quan hệ hành tinh được xác lập bởi vì việc này cho phép thể gây ra cho hiệu quả tinh tế những vấn đề đụng nhau giữa TriMesh với nhiều geoms khác nhau trong thời gian tồn tại của nó.

10.7.7 Geometry Transform class (Biến đổi hình học)

Một sự biến đổi hình học Transform viết tắt là 'T ' là một kiểu đối tượng hình học geom đặc biệt dùng để gói các geom ' E ' khác nhau, Cho phép E dễ ràng định vị và quay tùy ý đối vị trí tham khảo.

Đa số các geoms (hình cầu và hình hộp, …) có điểm tham khảo tương ứng với tọa độ trọng tâm của nọ, cho phép chúng sẽ dễ dàng nối tới những đối tượng động lực học. Những đối tượng **Transform** cho bạn điều chỉnh việc này linh hoạt hơn - ví dụ, bạn có thể dịch điểm tham khảo bằng cách offset tâm hình cầu, hoặc quay một hình trụ để trục của nó khác đi so với giá trị mặc định.

```
dGeomID dCreateGeomTransform (dSpaceID space);
```

Tạo ra một đối tượng biến đổi hình học mới, và trả lại chỉ số của nó. Nếu không gian space khác không, chèn nó vào trong không gian đó. Đối tượng hình học được gói là NULL.

```
void dGeomTransformSetGeom (dGeomID g, dGeomID obj);
```

Thiết lập đối tượng biến đổi hình `g` gói đối tượng hình học obj. Đối tượng `obj` không được chèn vào trong bất kỳ không gian nào, và không được liên hệ với bất kỳ đối tượng động lực học.

Nếu `g` có biến trạng thái `clear – up` đã bật , khi nó đã đóng gói một đối tượng, đối tượng cũ sẽ được phá hủy trước khi nó đóng gói đối tượng mới.

```
dGeomID dGeomTransformGetGeom (dGeomID g);
```

Hàm lấy đối tượng hình học mà đối tượng biến đổi hình học `g` gói.

```
void dGeomTransformSetCleanup (dGeomID g, int mode);
int dGeomTransformGetCleanup (dGeomID g);
```

Hàm đặt chế độ xóa đối tượng biến đổi hình học `g`. Nếu tham biến mode là 1, thì đối tượng được đóng gói sẽ bị hủy khi đối tượng biến đổi hình học bị hủy. Nếu mode là 0 điều này không xảy ra. Mode có giá trị mặc định là 0.

```
void dGeomTransformSetInfo (dGeomID g, int mode);
int dGeomTransformGetInfo (dGeomID g);
```

Hàm đặt và nhận "Thông tin" Kiểu biến đổi hình học g. Tham số mode có thể là 0 hoặc 1, nhưng mặc định là 0.

Với mode= 0, Khi một đối tượng biến đổi va chạm với đối tượng khác (Sử dụng hàm dCollide(tx geom,other geom,...)), thành phần g1 của cấu trúc dContactGeom được đặt là đối tượng hình học geom mà được đóng gói bởi đối tượng biến đổi đó. Giá trị của g1 cho phép người gọi thẩm vấn kiểu geom mà được biến đổi, nhưng nó không cho phép người gọi xác định vị trí trong những tọa độ toàn cầu hoặc đối tượng động lực học, trong khi cả hai thuộc tính này được sử dụng khác nhau cho geoms được đóng gói.

Với mode = 1, thành phần g1 của cấu trúc dContactGeom được đặt là bản thân đối tượng biến đổi. Điều này làm đối tượng có vẻ là giống như mọi loại geom khác, Trong khi dGeomGetBody() sẽ trả về đối tượng động lực học gắn với nó, và dGeomGetPosition() sẽ trả lại vị trí của nó trong tọa độ toàn cầu. Để làm chủ kiểu geom được gói trong thực tế trường hợp này hàm dGeomTransformGetGeom() phải đợc sử dụng.

10.8 Đối tượng người dùng tự định nghĩa (User defined classes)

Trong thư viện các đối tượng hình học được định nghĩa theo lớp viết theo cấu trúc C++. Nếu bạn muốn định nghĩa những lớp hình học của chính mình bạn có thể làm điều này theo hai cách :

1. Sử dụng những hàm C trong mục này. Điều này có lợi thế cung cấp một sự tách bạch giữa mã của thư viện và mã của bạn.

2. Thêm những lớp trực tiếp vào mã nguồn. Điều này có lợi thế là bạn có thể sử dụng C++ để viết code và kế thừa các lớp có trong thư viện, đây cũng là phương pháp ưa thích nếu lớp va chạm của bạn viết tốt hơn mã nguồn mở.

Sau đây là các hàm C API cho người dùng định nghĩa những lớp hình học.

Mỗi lớp hình học được người dùng định nghĩa có chỉ số là số nguyên duy nhất. Một lớp hình học mới (Gọi nó ' X ') Phải cung cấp cho thư viện những thông tin sau:

1. Các hàm điều khiển sinh các điểm tiếp xúc chạm giữa X và các lớp khác. Những hàm này phải có kiểu dColliderFn, Mà được định nghĩa như sau:

```
typedef int dColliderFn (dGeomID o1, dGeomID o2, int flags,
                         dContactGeom *contact, int skip);
```

Hàm này có giao diện như hàm dCollide(). Mỗi hàm sẽ điều khiển một trường hợp va chạm đặc biệt, nơi mà o1 có kiểu X Và o2 có các kiểu còn lại.

2. Hàm "chọn lựa" có kiểu dGetColliderFnFn, Mà được định nghĩa như sau:

```
typedef  dColliderFn * dGetColliderFnFn ( int num);
```

Hàm này có tham số là một số lớp (num), và trả lại va chạm khi X va chạm với lớp num. Nó trả về 0 nếu X không va với lớp num. Chú ý nếu có hai đối tượng X Và Y sẽ đụng nhau, duy nhất chỉ cần cung cấp một hàm va chạm giữa một đối tượng này với đối tượng khác.

3. Một hàm sẽ tính toán trục sắp hàng của đối tượng hình bao (AABB) của lớp mới này. Hàm này phải có kiểu dGetAABBFn, và được định nghĩa như sau:

```
typedef void dGetAABBFn (dGeomID g, dReal aabb[6]);
```

Hàm này cần tham số g, Là ID của kiểu X, Và trả về trục gióng hàng của hình bao đối tượng g. Mảng aabb có những phần tử (minx,maxx, miny, maxy, minz, maxz). Nếu bạn không muốn tính toán chặt hình bao AABB, Bạn có thể chỉ cần cung cấp một con trỏ tới hàm dInfiniteAABB(), để cho biết hình bao rộng mô tận.

4. Số bytes lưu dữ liệu lớp cũng phải cần đến. Ví dụ hình cầu cất giữ bán kính của nó trong vùng dữ liệu lớp, và hình hộp cất giữ cạnh lengths của nó ở đó.

Các hàm tương tác trên lớp hình học:

1. Phải có hàm hủy lớp. Đa số các lớp sẽ không cần hàm này, nhưng vài cũng có thể muốn thu hồi bộ nhớ để tiết kiệm cho các tài nguyên khác. Hàm này phải có kiểu dGeomDtorFn, và được định nghĩa như sau:

```
typedef void dGeomDtorFn (dGeomID o);
```

vói tham số o có kiểu X.

2. Một hàm để kiểm tra liệu có phải AABB đã cắt chéo nhau với AABB của X không. Hàm này được sử dụng như sự tiên đoán va chạm trước khi gọi hàm kiểm tra va chạm không gian. Hàm này phải có kiểu dAABBTestFn, Mà đợc định nghĩa như sau:

```
typedef int dAABBTestFn ( dGeomID o1, DGeomID o2, DReal aabb2[6]);
```

Vói tham số o1 có kiểu X. Nếu hàm này được gọi bởi dSpaceCollide() khi o1 cắt ngang geom o2, Mà có một AABB có các giá trị là aabb2 được lấy ra. Nó trả lại 1 nếu aabb2 cắt chéo nhau với o1, Hoặc 0 nếu không.

Hàm này rất là hữu ích, ví dụ, để giải quyết các bài toán có địa hình lớn có AABBs là rất lớn. Rõ ràng ta không thể kiểm tra sự giao nhau giữa địa hình với những đối tượng khác. Hàm này có thể kiểm tra

Các hàm quản lý những lớp tự định nghĩa :

```
int dCreateGeomClass (const dGeomClass *classptr);
```

Hàm này đăn ký một lớp hình học mới, định nghĩa bởi `classptr`. Chỉ só lớp mới được trả lại. Quy - ước sử dụng trong thư viện là gán số lớp cho một biến toàn cục với tên dXxxClass nơi Xxx là tên lớp (ví dụ là dSphereClass).

Sau đây là định nghĩa cấu trúc dGeomClass :

```
struct dGeomClass {
  int bytes;                    // bytes of custom data needed
  dGetColliderFnFn *collider;  // collider function
  dGetAABBFn *aabb;             // bounding box function
  dAABBTestFn *aabb_test;       // aabb tester, can be 0 for none
  dGeomDtorFn *dtor;            // destructor, can be 0 for none
};
```

```
void * dGeomGetClassData (dGeomID);
```

Tham số đầu vào là geom, Trả lại một con trỏ lưu dữ liệu người dùng.

```
dGeomID dCreateGeom (int classnum);
```

Tạo ra một geom có chỉ số lớp phải được cho vào. Dữ liệu người dùng thoạt tiên sẽ đặt là 0. Đối tượng này có thể được thêm vào trong không gian khi sử dụng hàm `dSpaceAdd()`.

Khi tạo một lớp mới bạn sẽ thường viết một trong các hàm sau đây :

1. Nếu lớp chưa được khởi tạo, thì khởi tạo ra nó. Bạn cần phải cẩn thận để chỉ tạo ra lớp một lần.

2. Gọi `dCreateGeom()` để tạo các đối tượng có kiểu mới.

3. Thiết lập dữ liệu người dùng.

10.9 Đối tượng đa hợp (Composite objects)

Ta hãy xem các đối tương sau:

• Một cái bàn được tạo ra bởi các hộp (miêu tả mặt bàn và các hộp miêu tả chân).

• Nhánh của cây mà được mô hình từ vài hình trụ giao.

Nếu những đối tượng này được nghĩa là vật cứng rắn thì thật cần thiết để sử dụng một thể rắn đơn để đại diện cho chúng.

Nhưng như thế rất khó để dò tìm va chạm, bởi vì không có lớp hình học đơn mà có thể đại diện một hình dạng phức tạp như một cái bàn hoặc nhánh cây. Giải pháp sẽ là sử dụng một đối tượng phức là sự kết hợp của vài geoms để khắc phục vấn đề này.

Không có hàm nào cần để quản lý những đối tượng phức hợp. Đơn giản là tạo ra mỗi thành phần geom và gắn nó tới cùng đối tượng động lực học. Để di chuyển và quay geoms riêng biệt tương đối với nhau trong cùng đối tượng ta sử dụng các đối tượng biến đổi hình học gói chúng lại.

Tuy nhiên có một cảnh báo là bạn cần phải không bao giờ tạo ra một đối tượng phức mà kết quả va chạm phát sinh những điểm tiếp xúc khá trùng nhau. Ví dụ, xem xét cái bàn được tạo ra bởi một cái hộp ở mặt bàn và bốn hình hộp cho những chân. Nếu những chân bàn rất ngắn, và cái bàn được đặt trên nền nhà, thì những điểm tiếp xúc phát sinh cho những cái hộp có thể trùng nơi những chân kết nối tới mặt bàn. Khi thực hiện tối u hóa những điểm tiếp xúc trùng hợp và hủy bỏ ra khỏi danh sách điều này có thể dẫn tới những lỗi và hành vi lạ. Trong ví dụ này cần chiếc bàn cần được điều chỉnh sao cho các hộp miêu tả chân bàn cần được dài thêm ra, để hủy bỏ các điểm tiếp xúc trùng nhau khi va chạm.

10.10 Hàm tiện tích

```
void dClosestLineSegmentPoints (const dVector3 a1, const dVector3 a2,
                                const dVector3 b1, const dVector3 b2,
                                dVector3 cp1, dVector3 cp2);
```

Tham số đầu vào là hai đoạn thẳng A và B với được giới hạn bởi hai điểm `a1 - a2` và `b1 - b2`, Hàm này trả về các điểm nằm trên A và B sao cho gần `cp1` và `cp2` nhất. Trong trường hợp những đoạn thẳng song song thì có nhiều giải pháp, một trong số đó là giải pháp lấy điểm đầu của ít nhất một đoạn thẳng. Điều này khắc phục trường hợp một trong hai đoạn có chiều dài length =0, Ví dụ như là `a1 = a2` và / hoặc `b1 = b2`.

```
int dBoxTouchesBox (const dVector3 _p1, const dMatrix3 R1,
                    const dVector3 side1, const dVector3 _p2,
                    const dMatrix3 R2, const dVector3 side2);
```

Hàm kiểm tra các hình hộp `(p1, R1,side1)` và `(p2, R2,side2)`, trả lại 1 nếu chúng cắt ngang nhau hoặc 0 nếu la không. p là tâm hộp, R là ma trận quay, và độ dài cạnh là thành phần của vectơ side1.

```
void dInfiniteAABB (dGeomID geom, dReal aabb[6]);
```

Hàm này có thể được sử dụng để thay thế AABB như một lớp hình học, nếu bạn không muốn tính toán chặt chẽ hình bao AABB. Hàm trả về + /- ∞ theo các hướng.

10.11 Chú ý khi mô phỏng cơ hệ lớn

10.11.1 Khi cơ hệ lớn.

Thông thường khi mô phỏng không gian hình học chứa đựng nhiều đối tượng trong số đó có một số đối tượng tĩnh (giá) ở đây các đối tượng này không được liên kết với đối tượng động lực học. Thư viện tối ưu việc tính va chạm bằng cách tìm ra các geoms không di chuyển và tính toán trước các thông tin để tiết kiệm thời gian.

10.11.2 Dùng các thư viện khác nhau về va chạm

Trong thư viện việc dò va chạm có thêm chức năng mở rộng cho phép thêm vào thư viện các hàm tìm va chạm của người lập trình miễn là nó có thể cung cấp những cấu trúc dContactGeom để khởi tạo tập điểm tiếp xúc.

Phần lõi động lực học của thư viện rất độc lập so với phần lõi dò va chạm, ngoại trừ bốn trường hợp sau:

1. Kiểu dGeomID phải được định nghĩa, trong khi mỗi đối tượng động lực học lưu giữ một con trỏ, trỏ tới đối tợng hình học đầu tiên được gắn.

2. Hàm dGeomMoved() phải được định nghĩa, sau đây là mẫu :

```
void dGeomMoved (dGeomID);
```

Hàm này được gọi trong phần mã động lực học khi một body di chuyển: Nó thông báo với phần hình học là đối tợng hình học có liên quan đến body bây giờ ở một vị trí mới.

3. Hàm dGeomGetBodyNext() phải được định nghĩa, sau đây là mẫu :

```
dGeomID dGeomGetBodyNext (dGeomID);
```

Hàm này được gọi trong phần mã động lực để duyệt qua danh sách geoms được gắn với body. Nó trả lại geom tiếp theo gán với body đó, hoặc 0 nếu không có nhiều một geom.

4. Hàm dGeomSetBody() phải được định nghĩa, sau đây là mẫu :

```
void dGeomSetBody ( dGeomID, dBodyID);
```

Hàm này được gọi trong đoạn mã hủy đối tượng động lực học (với tham số thứ hai dBodyID là 0) để ngắt tất cả các liên kết hình học với body.

Nếu bạn muốn thay thế các hàm dò va chạm trong thư viện – xin chú ý là bạn cần phải định nghĩa những kiểu hình học và những hàm này phù hợp.

Chương 11

CÁC CHÚ Ý KHI XÂY DỰNG CƠ HỆ

11.1 Độ chính xác và độ ổn định của cơ hệ

- Tích phân số không cho nghiệm chính xác
- Độ ổn định phụ thuộc nhiều yếu tố
- Cân nhắc giữa sự chính xác, độ ổn định của cơ hệ.

11.2 Ứng xử cơ hệ phụ thuộc vào bước tính (step size)

- Bước nhỏ hơn = chính xác, ổn định hơn hơn
- 10 * 0.1 kết quả khác với 5 * 0.2

11.3 Làm cho vật chuyển động nhanh hơn

Tốc độ thực hiện chương trình phụ thuộc vào nhân tố nào? Như ta biết mỗi khớp khi chuyển dịch trong không gian có số bậc tự do (DOFs) đặc trưng riệng. Ví dụ khớp cầu có số bậc tự do là 3 hạn chế 3, và khớp bản lề có bậc tự do là 1 bị hạn chế 5. Như vậy đối với mỗi vật được miêu tả bằng cách nối vơi nhau qua những nhóm khớp, khi đó ta có:

- m1 là số khớp nối trong nhóm,
- m2 là toàn bộ số bậc tự do của nhóm bị hạn chế, và
- n số đối tượng động lực học trong nhóm

Thì thời gian cần tính toán cho mỗi bước tính sẽ là :

$$k_1 O(m_1) + k_2 O(m_2^3) + k_2 O(n) \quad (11.1)$$

Hiện tại thư viện dụng một hệ thống ma trận có một hàng / cột cho mỗi DOF bị hạn chế . Trong một 10 body của một day xích được nối với nhau bằng khớp cầu, thì chiếm 30 - 40 % thời gian cho việc điền vào ma trận này, và 30 - 40 % cho các yếu tố khác.

Như vậy, tốc độ mô phỏng có thể xem xét trên các khía cạnh sau:

• Sử dụng ít nhất khớp nối - đối với các đối tượng động lực học có kích thước nhỏ có thể thay thế bằng các thạng thái động học thuần túy.

• Thay thế nhiều khớp bằng các khớp đơn giản hơn. Ở phần này để bạn phát huy hết khả năng chuyên ngành vào việc định các khớp chuyên dụng.

• Sử dụng ít khớp tiếp xúc.

11.4 Làm cho cơ hệ ổn định hơn

• Độ cứng của lò xo làm cơ hệ mất ổn định hơn.

• Ràng buộc cứng làm cơ hệ ổn định hơn.

• Độ ổn định phụ thuộc vào bước tính tích phân.

• Thiết lập giới hạn làm việc của các khớp, hạn chế dùng cơ cấu lò xo nếu có thể.

• Nếu vật di chuyển nhanh hơn bình thường hãy thay đổi bước tính cho hợp lý

Khi tăng giá trị của biến CFM toàn cục sẽ làm hệ thống ổn định hơn. Những ràng buộc thừa đấu tranh lẫn nhau và gây ra đổ vơ cơ hệ. Ví dụ nếu có khớp hai tiếp xúc phát sinh nối hai vật ở cùng một điểm hay một ví dụ khác là thay vì nối hai vật bằng một khớp bản lề ta lại dùng hai khớp cầu có tâm nằm trên trục khớp bản lề (Điều này là không tốt cho cơ hệ bởi vì hai khớp cầu hạn chế sáu bậc tự do, như khớp bản lề thực tế chỉ hạn chế năm bậc tự do).

Loại bỏ các ràng buộc thừa tránh sự tác động lẫn nhau của các vật nhưng vẫn miêu tả được quá trình vật lý của cơ hệ ta có thể dùng lực ràng buộc để thay thế. Ví dụ, ta muốn một vật bị ảnh hưởng bởi vật khác theo quy luật bay vòng quanh.

11.5 Tránh sự xuất hiện siêu liên kết

• Siêu liên kết xuất hiện khi có nhiều hơn một liên kết cần thiết để cưỡng ép chuyển động của vật.

• Có qua nhiều khớp nối giữa hai vật thể, đặc biệt các khớp tiếp xúc (không va chạm những phát sinh khớp tiếp xúc).

• Khi tăng giá trị của biến toàn cục CFM.

• Sử dụng tối thiểu số khớp để miêu tả hiện tượng vật lý mong muốn.

11.6 Các yếu tố khác

• Độ lẩy khi va chạm nếu quá mạnh hãy sử dụng đặc tính mềm của vật

• Giữ các chiều dài và những khối lợng có giá trị xung quanh 1

• LCP khi tính toán cần sử dụng rất nhiều vòng lặp. Nếu thời gian quá lâu, hãy tăng thêm giá trị của biến toàn cục CFM, hạn chế xuất hiện nhiều khớp sự tiếp xúc (hoặc tương tự).

• Đặt giới hạn làm việc của khớp bản lề khác + /- pi.

Chương 12

CÁC CÂU HỎI THƯỜNG GẶP

Chương này có vài câu hỏi chung và những câu trả lời của các chuyên gia. Cho bạn thông tin kỹ hơn về thư viện.

12.1 Làm thế nào nối một vật với giá?

Sử dụng hàm dJointAttach() với tham số (body, 0) hoặc (0, body).

12.2 Tại sao các vật bị nảy hoặc thâm nhập vào nhau khi va chạm? Kết quả nhận được là zero!

Trong một vài trường các vật rắn va chạm nhau nhưng không trả lại kết quả, chúng xuất hiện khi các vật thâm nhập vào nhau và được đẩy về một bên sao cho chúng chỉ chỉ cần chạm nhau. Vấn đề này có xấu hơn khi sử dụng bước tính lớn. Hiện tượng này là thế nào và quá trình xẩy ra ra sao?

Các ràng buộc tiếp xúc chỉ phát sinh khi phát hiện ra va chạm. Nếu trong một khoảng thời gian cố định tại bước tính hiện thời, các vật đã thâm nhập vào nhau thì hiện tượng nảy xẩy ra. Cơ chế giảm lỗi sẽ đẩy những thân thể về một bên, nhưng điều này cần một ít bước tính (phụ thuộc vào giá trị tham số ERP).

Sự thâm nhập của các vật thể và cơ chế giảm lỗi đẩy vật về một bên đôi khi làm những thân thể trông như chúng đang nảy lên, mặc dầu nó độc lập hoàn toàn. Liệu sự trả lại đúng hay sai.

Trong các thư viện khác đối với mỗi vật cần một biến lưu khoảng thời gian để kiểm tra trước để chắc chắn rằng những thân thể không bao giờ thâm nhập sâu vào nhau. Tuy nhiên trong các tính toán của thư viện này thời gian tính toán việc này là cố định, bằng cách tự động chọn một bước tính tối ưu nhất (Toàn bộ cấu trúc ARB phải tính toán kiểm tra sự thâm nhập đầu tiên, kết quả cho bước tính rất nhỏ).

Có ba cách sửa lỗi cho vấn đề này :

- Giảm bước tính tích phân thời gian thực.
- Tăng giá trị của biến ERP để khắc phục nếu có thể.
- Thay đổi bước tính tích phân ở các trạng thái khác nhau của cơ hệ

12.4 Cách thức tạo vật không di chuyển?

Hay cách nói khác là làm thế nào để vật không di chuyển, khi tương tác với những vật khác? Câu trả lời là chỉ tạo ra một đối tượng hình học tĩnh, không có gán cho nó một đối tượng động lực học nào. Nghĩa là Geom có liên hệ với một vật có chỉ số là zero. Sau đó khi hàm kiểm trava chạm giữa hai geoms có các chỉ số của vật khác không và zero, bạn có thể bỏ qua hai hai vật đó nhùn hàm dJointAttach() làm việc bình thường. Điều này tạo ra một sự tiếp xúc giữa thể rắn và giá.

Nếu ta gán cho vật một khối lượng lớn /mô men quán tính để mong muốn vật không chuyển động. Rồi đặt lại vị trí / hướng của vật sau mỗi bước tính điều này có thể gây ra đổ vỡ cơ hệ.

12.5 Vì sao lại phải đặt giá trị của ERP nhở hơn một?

Xuất phát từ định nghĩa của giá trị ERP, hình như cần đặt nó có giá trị bằng một là cách tốt nhất, bởi vì cần tất cả các lỗi sẽ hoàn toàn được sửa chữa tại mỗi bước tính. Tuy nhiên, thư viện sử dung nhiều sự xấp xỉ trong khi tính tích phân số, như vậy ERP = 1 sẽ không sửa 100 % lỗi. ERP = 1 có thể làm việc trong vài trường hợp, nhưng nó có thể cũng cho kết quả trong sự bất ổn định trong vài hệ thống. Trong những trường hợp này bạn nên giảm bớt giá trị của ERP để nhận được việc xử lý sự hệ thống tốt hơn.

12.6 Đặt trực tiếp vận tốc ban đầu cho vật, hay đặt lực hoặc mô men lực?

Bạn đặt trực tiếp lên vật những vận tốc nếu bạn đang đặt cấu hình ban đầu cho một cơ hệ nào đó. Nếu bạn đặt vận tốc vào vật trong mỗi bước tính thì có lẽ bạn đang lạm dụng mô hình vật lý, nghĩa là bắt buộc hệ thống làm cái mà bạn muốn hơn là để cho nó xảy ra tự nhiên.

Phương pháp được ưa thích việc cho vật một vận tốc ban đầu khi mô phỏng ta nên sử dụng những khớp nối mô tơ. Chúng có thể đặt cho vật có một vân tốc mong muốn trong mỗi bước tính, miễn là lực / mô mem đủ lớn.

12.7 Tại sao, khi đặt vận tốc trực tiếp cho vật, thì tốc độ xử lý chậm hơn khi nối vật nối tới những thân thể khác?

Điều này có gì nghĩa là bạn đặt vận tốc cho một vật mà không đặt cùng vận tốc cho các vật được nối với nó. Khi bạn làm điều này, bạn gây ra lỗi hệ thống trong bước tính kế tiếp làm các vật tách rời khỏi khớp nối. Cơ chế giảm lỗi sẽ dần dần sửa chữa cho điều này và kéo những vật dọc theo trục khớp, nhưng nó có thể mất một ít thời gian để tính toán. Đây chính là lý do làm cho vật bị kéo lê khỏi điểm gốc của vật.

Đặt vận tốc cho một vật sẽ ảnh hưởng một mình vật đó. Nếu nó được nối tới những vật khác, bạn phải đặt vận tốccho mỗi một vật một cách riêng rẽ để ngăn ngừa hành vi này.

12.8 Tôi có nên đặt đơn vị tỉ lệ xấp xỉ bằng 1 không?

Những độ dài lengths và khối lượng nhỏ sẽ làm việc không vấn đề gì nào. Tuy nhiên thỉnh thoảng bạn có thể trải qua những vấn đề ổn định mà gây ra bởi sự thiếu sự chính xác trong các hệ số. Nếu đây là nguyên nhân, bạn có thể thử để tỉ lệ lengths và những khối lượng của vật trong cơ hệ xung quanh 0.1..1.0. Bước tính cần phải lấy lại tỉ lệ tương ứng. Nguyên tắc này cũng được sử dụng với những lengths và khối lượng lớn.

12.9 Tôi mô phỏng xe ô tô, nhưng các bánh xe không đáp ứng chức năng!

Nếu bạn đang xây dựng một cơ hệ mô phỏng ô tô, cách chung nhất là bạn tạo ra một thân thể khung gầm và bốn vật khác là các bánh xe.

Tuy nhiên, bạn có thể phát hiện ra rằng khi bạn lái xe quay vòng những bánh xe quay sai hướng, cứ như thể các khớp nối không có tác dụng. Vấn đề này càng rõ khi quan sát ô tô chuyển động nhanh (nghĩa là các bánh xe đang quay nhanh), và ô tô thử quay một góc. Những bánh xe quay ra khỏi những sự ràng buộc thích hợp của nó cứ như thể trục bánh xe bị chúi xuống. Nếu những báng xe quay chậm, hoặc việc quay được làm chậm, thì vấn đề này ít xuất hiện hơn.

Vấn đề là những lỗi khi tích phân gây ra khi bánh xe quay ở tốc độ. Có hai hàm để sửa lỗi này là: dBodySetFiniteRotationMode() và dBodySetFiniteRotationAxis(). Những bánh xe cần phải được hạn chế bởi tốc độ quay giới hạn, và các trục quay của khớp bản lề cần được giới hạn tại mỗi bước tính. Hai hàm này sửa chữa hầu hết các lỗi.

12.10 Tôi làm như thế nào để có tương tác va chạm một chiều

Điều này bạn phải có hai vật (A và B) đụng nhau. Sự chuyển động của A ảnh hưởng đế sự huyển động của B, nhưng B ảnh hưởng đến A. Điều này có thể cần thiết, ví dụ, nếu B là một camera được mô phỏng về mặt vật lý trong môi trường VR. Camera cần sự đáp lại va chạm để nó không đi vào trong bất kỳ những đối tượng trong cảnh mô phỏng do tính toán nhầm, sự chuyển động sai của camera không cần ảnh hưởng đến sự mô phỏng. Điều này cần giải quyết như thế nào?

Sau đây là một giải pháp khá tốt : Khi va chạm được phát hiện ra, hãy không pháp sinh các điểm tiếp xúc giữa A và B như va chạm thông thường. Thay vào đó, hãy gán sự tiếp xúc chung giữa B và giá.

12.12 Vật quay không ổn định!

Nếu bạn có một hình hộp với những cạnh có độ đài khác nhau, cho nó quay trong không gian tự do, bạn quan sát thấy rằng nó ngã cùng tốc độ. Nhưng đôi khi trong thư viện hình hộp sẽ quay với tốc độ đặc trưng của chính nó, quá trình này nhanh hơn và nhanh hơn cho đến khi nó " Nổ tung " (Biến mất vàokhông gian vô hạn). Sau đây là giải thích :

Thư viện sử dụng một máy tích phân nửa ẩn cấp một. " Semi ẩn " Có nghĩa rằng vài lực lọng đọc tính toán cứ nh thể một máy tích phân ẩn đang đọc sử dụng, và (kẻ) khác bắt buộc đọc tính toán cứ nh thể máy tích phân (thì) rõ ràng. Sự ràng buộc bắt buộc (đọc áp dụng tới những thân thể để giữ những sự ràng buộc cùng nhau) (thì) ẩn, Và

" ngoài " những lực lọng (áp dụng bởi ngời dùng, và vì quay vòng những hiệu ứng) (thì) rõ ràng. Bây giờ, sự không chính xác trong những máy tích phân ẩn đọc biểu lộ nh một sự giảm trong năng lọng - trong những từ khác máy tích phân giấp ót hệ thống cho bạn. Sự Không chính xác trong những máy tích phân rõ ràng có hiệu ứng đối diện - nó tăng thêm năng lọng hệ thống. (Cái) này tại sao là những hệ thống đóng vai với cấp một rõ ràng (mà) những máy tích phân có thể nổ tung.

Nh vậy, một thân thể đơn ngã trong không gian (thì) tích hợp rõ ràng có hiệu quả. Nếu là thân thể ' Những chốc lát s (của) quán tính (thì) bằng nhau (E.g. nếu nó là một hình cầu) rồi trục quay sẽ còn lại không thay đổi, và lỗi máy tích phân sẽ (thì) nhỏ. Nếu là thân thể ' Những chốc lát s (của) quán tính (thì) không bằng nhau rồi trục quay lắc l nh động lọng đọc chuyển giữa những phong hớng quay khác nhau. Đây là hành vi vật lý đúng, nhng nó kết quả trong lỗi máy tích phân cao hơn. Máy tích phân trong tròng hợp này (thì) rõ ràng sao cho lỗi tăng thêm năng lọng , mà gây ra sự quay nhanh hơn và nhanh hơn, gây ra càng ngày càng dẫn dắt lỗi cho sự bùng nổ. Vấn đề (thì) hiển nhiên đặc biệt với những đối tọng mỏng dài (lâu), nơi mà 3 chốc lát (của) quán tính (thì) không bằng nhau cao.

Để ngăn ngừa điều này, làm một hoặc hơn (của) sau đây :

• chắc chắn rằng tùy thích những thân thể quay (thì) cân đối động (i.e. tất cả các chốc lát (của) quán tính là cũng nh thế

- ma trận quán tính là một thời báo không thay đổi ma trận nhận biết). Chú ý rằng bạn có thể vẫn còn trả lại và đụng nhau với một cái hộp mỏng dài (lâu) dù nó có quán tính (của) một hình cầu đi nữa.

• chắc chắn rằng tùy thích những thân thể quay don ' t quay nhanh quá (e.g. don ' t áp dụng những lực xoắn lớn, hoặc cung cấp sự làm nhụt thêm bắt buộc).

• thêm những phần tử làm nhụt Thêm vào môi trờng, e.g. don ' t sử dụng những sự va chạm bouncy mà có thể phản chiếu năng lợng.

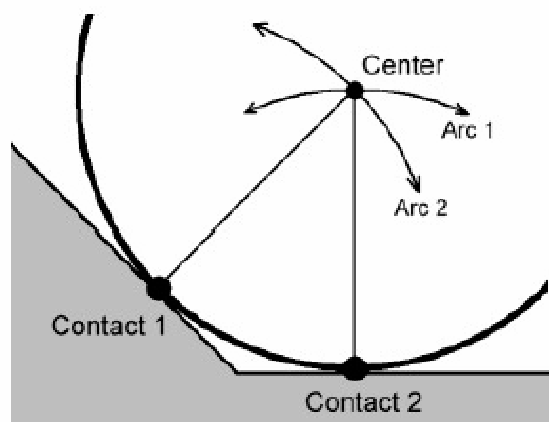
• Sử dụng timesteps nhỏ hơn. Đây xấu cho hai reasons: Nó ' S chậm hơn, và Thơ ca ngợi hiện thời chỉ có một máy tích phân cấp một sao cho sự chính xác đợc thêm là cực tiểu.

• Sử dụng một máy tích phân thứ tự cao hơn. Đây còn cha là một tùy chọn trong Thơ ca ngợi.

Trong tơng lai Tôi có thể thêm một đặc tính vào Thơ ca ngợi để sửa đổi động lực học quay vòng (của) những thân thể đợc lựa chọn để chúng không trng bày lỗi quay vòng với máy tích phân những Thơ ca ngợi.

12.13 Khi các vật quay đối khi bị kẹt giữa các geoms

Khi một vật lăn trên một có nhiều đối tượng hình học. Ví dụ, một ô tô điều khiển trên một địa hình (Các vật lăn là các bánh xe). Ta thấy các vật này dừng lại một cách đầy bí ẩn khi chúng lăn qua vật này sang vật khác, hoặc khi chúng nhận nhiều điểm tiếp xúc để mô tả lực ma. Mục này giải thích vấn đề và đưa ra giải pháp.



Hình 12.1: Vấn tiếp xúc khi lăn.

12.13.1 Các vấn đề

Một ví dụ miêu tả hệ thống vào trạng thái như vậy Hình 12.1, thể hiện mộtquả bóng cuộn xuống một dốc và chạm nền.

Bình thường, quả bóng cần phải tiếp tục lăn dọc theo nền, sang bên phải. Tuy nhiên, nếu thư viện sử dụng mạc định kiểu tiếp xúc ma sát điều này làm quả bóng sẽ dừng lại khi nó chạm nền. Tại sao?

Trong thư viện có hai cách tính xấp xỉ ma sát : cách mặc định (Ma sát hộp) và một cách đã cải tiến (Gọi "Sự xấp xỉ hình chóp ma sát 1 ") các điểm tiếp xúc thu được bởi cờ dContactApprox1 đặt trong các kiểu tiếp xúc bề mặt.

Xem xét trên hình 12.1. Có hai điểm tiếp xúc, một giữa quả bóng và dốc, điểm khác giữa quả bóng và nền. Nếu kiểu ma sát hộp được sử dụng tại cả hai điểm tiếp xúc và tham số mu được đặt bằng dInfinity thì quả bóng không thể trượt ra khỏi dốc hoặc nền ở mọi sự tiếp xúc.

Nếu không có hienẹ tượng trượt tại điểm tiếp xúc quả bóng, thì tâm quả bóng phải di chuyển dọc theo một đường dẫn mà là cung xung quanh điểm tiếp xúc. Như vậy tâm quả bóng được yêu cầu đồng thời di chuyển dọc theo đường dẫn là " Arc 1 " và " Arc 2 ". Cách duy nhất để thỏa mãn cả hai đường dẫn tại cùng một thời điểm là quả bóng dừng việc di chuyển.

Đây không phải là một lỗi trong thư viện. Nhưng điều gì đang xẩy ra ở đây? Những đối tượng trong thực tế bị kẹt như thế này. Vấn đề là khi sử dụng mô hình ma sát đơn giản "Ma sát hộp" Sự xấp xỉ thành phần lực ma sát tiếp tuyến luôn làm cho các ràng buộc tiếp xúc dừng một cách độc lập ngăn ngừa sự thâm nhập. Đây không phải là hiện tượng vật lý trong thực tế, vì vậy chúng ta không cần ngạc nhiên vì rằng chuyển động thực tế không cho kết quả như vậy.

Hãy chú ý rằng vấn đề này không xuất hiện nếu mu đặt bằng 0, nhưng đây không phải là một giải pháp có ích bởi vì chúng ta cần có lực ma sát để mô phỏng sát với thực tế.

12.13.2 Cách giải quyết

Giải pháp sẽ sử dụng cờ dContactApprox1 trong các tiếp xúc bề mặt , và đặt giá trị mu cho thích hợp giữa 0 và ∞. Điều này bảo đảm rằng sẽ chỉ có một thành phần lực ma sát tiếp tuyến chống trượt tại điểm tiếp xúc nếu thành phần pháp tuyến khác 0. Trong ví dụ trên lực này quay ra ngoài tại điểm tiếp xúc contact 1 lực ma sát pháp tuyến bằng 0. Vì vậy sẽ không có lực tác động vào điểm tiếp xúc contact 1, và vấn đề được giải quyết! (quả bóng sẽ cuộn dọc theo nền .)

Kiểu dContactApprox1 có thể không (thì) thích hợp trong tất cả các tình trạng, mà tại sao là nó (thì) để chọn. Đó là quan trọng mà nhớ cái đó, mặc dầu Nó là một sự xấp xỉ ma sát tốt hơn, Nó (thì) không phải là ma sát Culông thật.

Nh vậy thật là vẫn còn có thể rằng bạn có thể gặp vài ví dụ (của) hành vi không vật lý.