Kỷ yếu Hội nghị KHCN Quốc gia lần thứ XII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR); Huế, ngày 07-08/6/2019
DOI: 10.15625/vap.2019.00031

KỸ THUẬT PHÁT HIỆN NHANH VA CHẠM CỦA VẢI TRONG THỰC TẠI ẢO SỬ DỤNG PHƯƠNG PHÁP TÍNH TOÁN SONG SONG

Nghiêm Văn Hưng[1,3], Đặng Văn Đức[2], Trịnh Hiền Anh[2], Hoàng Việt Long[3], Nguyễn Văn Căn[3]

[1]Học viện Khoa học và Công nghệ, Viện Hàn lâm Khoa học và Công nghệ Việt Nam
[2]Viện Công nghệ thông tin, Viện Hàn lâm Khoa học và Công nghệ Việt Nam
[3]Trường Đại học Kỹ thuật - Hậu cần Công an nhân dân, Bộ Công an

nghiemvanhung1985@gmail.com, dvduc@ioit.ac.vn, hienanh@ioit.ac.vn,
longhv08@gmail.com, nguyenvancan@gmail.com

TÓM TẮT: *Phát hiện va chạm của vải trong môi trường 3D là một trong những tác vụ phức tạp của hệ thống thực tại ảo. Sự tương tác tự nhiên của vải với các vật thể khác và với chính nó như gập, cuộn,... làm cho việc phát hiện va chạm trở nên khó khăn hơn. Trong bài báo này, chúng tôi trình bày một kỹ thuật tính toán song song để tăng tốc phát hiện va chạm của mô hình vải trên hệ thống CPU đa nhân, cách tiếp cận dựa trên phân tách vùng kiểm tra va chạm và giảm số lượng phép kiểm tra cơ sở. Kỹ thuật đề xuất đã được cài đặt bằng ngôn ngữ C++ trong bộ Visual Studio 2010 với OpenMP trên hệ thống máy tính sử dụng CPU Intel Xeon Core E5-2665 (8 nhân) và tỏ ra hiệu quả với ba bộ dữ liệu mô hình vải trong thư viện mở GAMMA cho phép tăng nhanh tốc độ phát hiện va chạm gấp 6.4, 6.68 và 7.7 lần so với kỹ thuật ICCD của Tang và cộng sự.*

Từ khóa: *Phát hiện va chạm, tự va chạm, mô hình vải, tính toán song song.*

I. GIỚI THIỆU

Phát hiện va chạm (Collision Detection) là một trong những tác vụ cơ bản của các hệ thống mô phỏng thực tại ảo, đồ họa máy tính, điều khiển robotics, games,... Các đối tượng 3D trong hệ thống mô phỏng có thể là các mô hình vật thể rắn (bức tường, thanh gỗ,…) hoặc mô hình vật thể biến dạng (vải, tóc,...), trong đó quá trình tự va chạm (Self Collision) thường xảy ra trong các mô phỏng mô hình vật thể biến dạng. Tự va chạm là sự va chạm giữa các thành phần trong cùng một mô hình 3D [23]. Hình 1 minh họa quá trình va chạm diễn ra trên chiếc váy của một cô gái đang múa, các phần vải tự va chạm đã được đánh dấu bằng màu sáng. Đã có nhiều công bố trong những năm gần đây cải tiến kỹ thuật phát hiện va chạm trong các mô hình vật thể biến dạng [18, 22, 23, 30], hầu hết trong số đó đều dựa trên cấu trúc hệ bao (Bounding Volume Hierarchies - BVHs). Đối với các mô hình vật thể biến dạng như vải thì chi phí duyệt BVHs phụ thuộc vào các cấu hình biến dạng, vấn đề đặt ra là cần thiết kế thuật toán có thể hoạt động tốt trên các cấu hình biến dạng khác nhau và kỹ thuật tính toán song song có thể giải quyết vấn đề đó. Trong bài báo này, chúng tôi đề xuất kỹ thuật tăng tốc phát hiện va chạm của mô hình vải trên nền kiến trúc bộ xử lý đa nhân (Multi-core CPU). Kết quả nghiên cứu có thể áp dụng cho các hệ thống đồ họa mô phỏng nhân vật có tương tác với chất liệu vải như: Thiết kế quần áo ảo, trình diễn thời trang ảo,...

Hình 1. Các phần vải tự va chạm (đã được đánh dấu bằng màu sáng) trên chiếc váy của một cô gái đang múa

Kế thừa và phát triển từ công bố của Tang và cộng sự [25] về va chạm trong các mô hình vật thể biến dạng, kỹ thuật đề xuất trong bài báo này cho phép tăng nhanh tốc độ phát hiện va chạm trong các mô hình vải khác nhau thuộc bộ dữ liệu của thư viện mở GAMMA (Geometric Algorithms for Modeling, Motion, and Animation) của Trường Đại học Bắc Carolina, Mỹ.

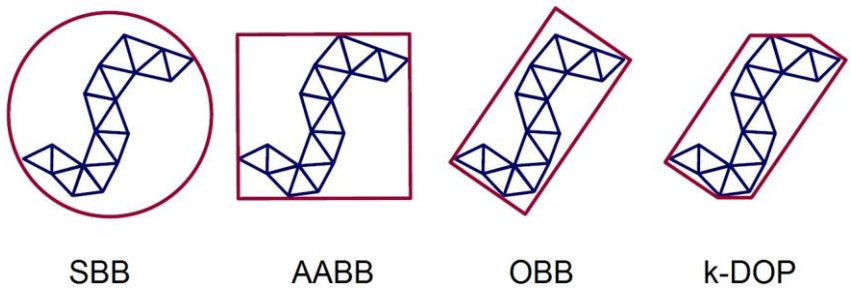
Phần còn lại của bài báo được cấu trúc như sau: Mục II trình bày về bài toán phát hiện va chạm và các nghiên cứu liên quan. Mục III trình bày cơ sở lý thuyết, cách tiếp cận song song và kỹ thuật đề xuất. Kết quả thử nghiệm được trình bày trong mục IV. Mục V phân tích đánh giá và mục VI là kết luận và định hướng nghiên cứu tiếp theo.

II. BÀI TOÁN PHÁT HIỆN VA CHẠM VÀ CÁC NGHIÊN CỨU LIÊN QUAN

Bài toán phát hiện va chạm trong môi trường thực tại ảo được phát biểu như sau: Cho n đối tượng $\{O_0, O_1, \dots, O_{n-1}\}$, phát hiện va chạm là quá trình xác định xem các đối tượng có giao nhau hay không, tức là kiểm tra $O_i \cap O_j, \forall i \neq j,\ i = 0, \dots, n-1, j = 0, \dots, n-1$ và $n \in \mathbb{N}$. Trong đó, $O_i = \{p_0^i, p_1^i, \dots, p_k^i\}$ là một tập gồm k mặt cơ sở

(primitives) và $O_j = \{p_0^j, p_1^j, \dots, p_m^j\}$ là một tập gồm m mặt cơ sở, va chạm của O_i và O_j xảy ra khi $p_a^i \cap p_b^j \neq \emptyset$ với $a = 0, \dots, k,\ b = 0, \dots, m$ và $k, m \in \mathbb{N}$. Nếu xem xét trường hợp tự va chạm của O_i thì cần kiểm tra $p_a^i \cap p_b^i$ với $a, b = 0, \dots, k$. Các mặt cơ sở thường được sử dụng là các tam giác. Kết quả trả về của bài toán phát hiện va chạm ở dạng Có/Không (Boolean), nếu có va chạm xảy ra thì cần tính toán các điểm tiếp xúc và thời điểm tiếp xúc (Time of contact). Việc phát hiện va chạm của các mô hình vật thể biến dạng (điển hình là mô hình vải) được đánh giá là phức tạp hơn việc phát hiện va chạm của các mô hình vật thể rắn vì phải xem xét đến quá trình biến dạng và vấn đề tự va chạm.

Kỹ thuật phát hiện va chạm sử dụng cấu trúc hệ bao (Bounding Volume Hierarchies - BVHs): Có thể tiến hành kiểm tra từng mặt của đối tượng này có giao cắt với một mặt nào đó của đối tượng kia hay không để phát hiện va chạm. Tuy nhiên mỗi đối tượng 3D được tạo thành từ rất nhiều mặt nên chi phí để kiểm tra giao nhau của các cặp mặt rất lớn. Hầu hết các hệ thống mô phỏng thực tại ảo đều sử dụng phương pháp phát hiện va chạm sử dụng khối hình học bao quanh đối tượng (khối bao). Việc phát hiện va chạm của các đối tượng được đưa về bài toán phát hiện giao nhau giữa các khối bao. Hình 2 mô tả một số dạng khối bao thường dùng trong các kỹ thuật phát hiện va chạm: Khối bao theo các cạnh song song với các trục tọa độ (Axis Aligned Bounding Box - AABB) [2], khối bao dạng hình cầu (Sphere Bounding Box - SBB) [3], khối bao theo hướng của đối tượng (Oriented Bounding Box - OBB) [9] và khối bao đa diện rời rạc có hướng (k-Discrete Oriented Polytopes - k-DOP) [12]. BVHs là một cấu trúc dữ liệu dạng cây được tạo thành trên cơ sở phân tích các đối tượng cần được kiểm tra va chạm. BVHs đóng vai trò quan trọng trong việc biểu diễn các vật thể và cho phép giải quyết nhiều vấn đề trong phát hiện va chạm [4, 5, 7, 16, 29]. Với các mô hình vật thể biến dạng như vải thì BVHs cần phải được dựng lại sau mỗi biến dạng, giải pháp được đề xuất là áp dụng kỹ thuật tái cấu trúc có chọn lọc [15], tuy nhiên chi phí để tái cấu trúc là rất lớn.



Hình 2. Một số dạng khối bao được sử dụng trong các kỹ thuật phát hiện va chạm

Kỹ thuật phát hiện va chạm liên tục và phát hiện va chạm tăng cường: Nhiều thuật toán hiệu quả đã được cải tiến để phát hiện va chạm liên tục giữa các mô hình vật thể rắn [33], mô hình khớp nối [34] và các mô hình vật thể biến dạng [25]. Kỹ thuật phát hiện va chạm liên tục áp dụng phương pháp nội suy tuyến tính giữa các đỉnh của mô hình, thực hiện các phép kiểm tra cơ sở giữa các cặp tam giác và tính toán thời điểm tiếp xúc. Kỹ thuật phát hiện va chạm tăng cường: Kỹ thuật này phát hiện va chạm nhanh trong các ứng dụng tương tác, cách tiếp cận chính là thực hiện các tính toán va chạm tăng cường giữa các khung (frames), áp dụng cho các đa giác lồi [8]. Một số thuật toán duy trì cấu trúc phân cấp để truy vấn nhanh hơn [12]. Hầu hết các kỹ thuật này đã được sử dụng để tăng tốc kiểm tra va chạm giữa các mô hình vật thể rắn. Các thuật toán dựa trên sự kiện [33] và động học [31] được đề xuất cho các mô hình vật thể biến dạng.

Kỹ thuật phát hiện va chạm song song: Kỹ thuật này khai thác khả năng xử lý song song của CPU [28] hoặc GPU [23] để tăng tốc phát hiện va chạm. Thao tác chủ yếu để kiểm tra va chạm là phép duyệt trên cấu trúc cây BVTT (Bounding Volume Traversal Tree) [17]. Rao và cộng sự [19] đã chỉ ra rằng hiệu quả của phép duyệt theo chiều sâu song song phụ thuộc vào sơ đồ phân phối các tiến trình và cấu trúc hệ bao. Các kỹ thuật cân bằng tải khác nhau cũng được đề xuất trong [13]. Reinefeld [21] đã so sánh các chiến lược cân bằng tải của các phương pháp duyệt theo chiều sâu để tăng tốc phát hiện va chạm. Assarsson [1] đã tăng tốc phát hiện va chạm giữa các mô hình vật thể rắn CAD nhanh hơn ba lần trên CPU 8 nhân. Grinberg [10] đã đề xuất một phương pháp song song phân vùng tác vụ tĩnh để kiểm tra va chạm. Chen và cộng sự [6] đã chứng minh ưu điểm của việc sử dụng các kiến trúc CPU đa nhân để phát hiện va chạm và mô phỏng vật lý. Các thuật toán phát hiện va chạm trong mô phỏng chất liệu vải đã được đề xuất cho các kiến trúc bộ nhớ phân tán [27]. Các kỹ thuật đa luồng được sử dụng trong [26] để tăng tốc xử lý va chạm cho mô phỏng chất liệu vải. Một kỹ thuật phân tách tác vụ được sử dụng trong [11] để tính toán trên kiến trúc CPU đa nhân. Kỹ thuật được sử dụng trong bài báo này là phát hiện va chạm liên tục tính toán song song trên CPU đa nhân.

III. KỸ THUẬT PHÁT HIỆN VA CHẠM CỦA MÔ HÌNH VẢI

A. Cơ sở lý thuyết

1. Cấu trúc dữ liệu

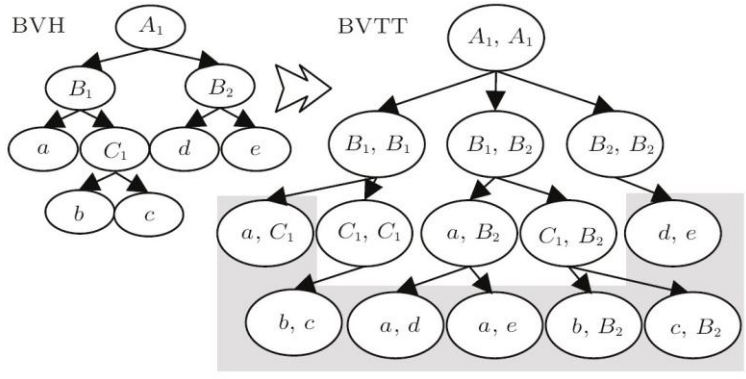
Khối bao đa diện rời rạc có hướng (k-Discrete Oriented Polytopes - k-DOP) là một đa giác lồi chứa đối tượng được xây dựng bằng cách lấy một số mặt phẳng định hướng phù hợp ở vô cực và di chuyển chúng cho đến khi chúng va chạm với vật thể, giao điểm của nửa không gian giới hạn bởi các mặt phẳng. Các giá trị của k được lựa chọn để xây dựng khối bao [12] là 6, 14, 18, 26. Việc tính toán cho từng giá trị của k được trình bày chi tiết trong phần C mục III.

Hệ bao BVH (Bounding Volume Hierarchies) là một cấu trúc dữ liệu dạng cây được tạo thành trên cơ sở phân tích các đối tượng cần được kiểm tra va chạm. BVHs đóng vai trò quan trọng trong việc biểu diễn các vật thể và cho phép giải quyết nhiều vấn đề trong phát hiện va chạm [4, 5, 7, 16, 29]. Với các mô hình vật thể biến dạng như vải thì BVH cần phải được dựng lại sau mỗi biến dạng, giải pháp được đề xuất là áp dụng kỹ thuật tái cấu trúc có chọn lọc [15]. Quá trình kiểm tra va chạm bắt đầu duyệt từ nút gốc của BVH.

Cấu trúc cây BVTT (Bounding Volume Traversal Tree) biểu diễn hệ thống phân cấp của các phép kiểm tra chồng lấp (testing for overlap) được thực hiện trong quá trình duyệt. BVTT được sử dụng trong thuật toán phát hiện va chạm [8, 14, 17, 31]. Mỗi nút trong BVTT đại diện cho một phép kiểm tra chồng lấp duy nhất giữa một cặp khối bao (BV) hoặc kiểm tra tự va chạm trên một trong các BV. BVTT không còn là cây nhị phân và một số nút bên trong của cây BVTT có thể có ba nút con.

Vùng kiểm tra BVTT: Vùng kiểm tra của BVTT là một tập hợp các nút bên trong và nút lá (nơi phép duyệt kết thúc trong khi thực hiện truy vấn va chạm trong một thời gian nhất định). Vùng kiểm tra BVTT phản ánh phép duyệt truy vấn va chạm.

Hình 3 minh họa ví dụ về BVH, BVTT tương ứng và vùng kiểm tra BVTT. Mô hình vải được tổ chức thành một BVH là cây có gốc (A_1). BVTT tương ứng là cây có gốc (A_1, A_1) và vùng kiểm tra của BVTT được tô bằng màu xám.



Hình 3. Ví dụ minh họa BVH, BVTT và vùng kiểm tra của BVTT

Việc phát hiện va chạm trên các mô hình vải tập trung vào hai vấn đề chủ yếu là các phép kiểm tra cơ sở và tự va chạm. Các phép kiểm tra cơ sở: Thực hiện 15 phép kiểm tra cơ sở giữa các mặt (Face), cạnh (Edge) và đỉnh (Vertex) của hai tam giác: Gồm 6 phép kiểm tra đỉnh-mặt (Vertex-Face) và 9 phép kiểm tra cạnh-cạnh (Edge-Edge) [18]. Tự va chạm: Do chuyển động biến dạng và thay đổi hình dạng có thể dẫn đến tự va chạm, đặc điểm của các tam giác liền kề trong lưới cần phải được xem xét.

2. Tính toán phân cấp song song (Parallel Hierarchical Computation - PHC)

Tang và cộng sự [25] đã mô tả phương pháp phát hiện va chạm dựa trên tính toán phân cấp tuần tự bao gồm bốn pha: Cập nhật BVH; Tính toán và duyệt BVTT; Kiểm tra các cặp tam giác không liền kề; Kiểm tra các cặp tam giác liền kề. Một tập rút gọn là tập con của tập các mặt, cạnh và đỉnh thuộc các tam giác liền kề không được kiểm tra trong quá trình xử lý các cặp tam giác không liền kề. Bằng cách chỉ tiến hành kiểm tra trên tập rút gọn, số lượng phép kiểm tra cơ sở cho các tam giác liền kề có thể giảm xuống.

Việc cập nhật và xử lý cặp không liền kề của BVH có thể thực hiện song song trên nhiều nhân. Nhưng việc song song hóa tính toán BVTT tương đối khó khăn. Do đó, khó khăn đối với việc phát hiện va chạm song song là đưa ra các kỹ thuật song song thích hợp để duyệt BVTT và tính toán các cặp có khả năng chồng lấp [11, 26, 27]. Trong pha thứ hai cần duyệt qua BVTT theo chiều rộng, thu thập tất cả thông tin các tác vụ phụ truy vấn va chạm và đưa chúng vào một ngăn xếp. Sau đó, tất cả các tác vụ phụ trong ngăn xếp được thực thi song song trên các nhân của CPU.

Tuy nhiên, cách tiếp cận PHC có điểm hạn chế là do tính chất động của các mô hình vải gây khó khăn trong việc phân phối tải của truy vấn va chạm giữa hai nút BVH. Phần tiếp theo trình bày kỹ thuật phân tách dựa trên vùng kiểm tra để phát hiện va chạm nhanh hơn PHC.

B. Phát hiện va chạm song song dựa trên phân tách vùng kiểm tra BVTT

1. Song song hóa BVTT cho các mô hình vải

Ý tưởng song song hóa kỹ thuật phát hiện va chạm chủ yếu tập trung vào phép duyệt BVTT [10]. Đối với vấn đề này, các chiến lược phân bổ tác vụ song song đã được áp dụng: Phân bổ động và phân bổ tĩnh, cả hai đều nhằm mục đích cân bằng tải tính toán. Chiến lược phân bổ động được thực hiện bằng cách phân phối ngẫu nhiên tải tính toán cho bộ xử lý hoặc luồng trong quá trình phát hiện [27]. Theo hướng này, chúng tôi đề xuất sử dụng kỹ thuật phân tách dựa trên vùng kiểm tra. Vùng kiểm tra của BVTT là một tập hợp các nút bên trong và nút lá (nơi phép duyệt kết thúc trong khi thực hiện truy vấn va chạm trong một thời gian nhất định). Vùng kiểm tra phản ánh số lượng cây được duyệt qua cho mỗi trường hợp của truy vấn va chạm. Tuy nhiên, phép duyệt BVTT là không thể tránh khỏi việc cập nhật vùng kiểm tra và quá trình duyệt này khó có thể được thực hiện cho mọi khung hình. Trong khi đó, chiến lược phân bổ tĩnh phân phối tải trước khi quá trình phát hiện bắt đầu [10], hạn chế chính của nó là không thể đảm bảo tải tính toán cân bằng giữa các luồng hoặc bộ xử lý.

2. Xây dựng và cập nhật BVTT

Cấu trúc BVTT được xây dựng theo phương pháp từ trên xuống (Top-down). Nhưng trong quy trình mô phỏng, các mô hình vải được chuyển đổi từ khung này sang khung khác. Các BVTT của chúng phải được cập nhật để phát hiện va chạm. Cách tiếp cận của chúng tôi giải quyết vấn đề này theo phương pháp từ dưới lên (Bottom-up) hiệu quả hơn là phương pháp từ trên xuống. Trong mỗi khung, đối với mỗi nút lá, khối bao được cập nhật. Do đó, các nút cha được cập nhật là ôm khít tất cả các khối con của nó. Quá trình kết thúc cho đến khi cập nhật nút mái hoàn thành.

C. Phân tích

Đối với thuật toán tuần tự trong [25], chi phí tính toán T_s của nó được tính theo công thức:

$$T_s = C_t + C_b + C_e \tag{1}$$

trong đó C_t là chi phí duyệt BVTT; C_b là chi phí của các phép kiểm tra chồng lấp khối bao; C_e là chi phí của các phép kiểm tra cơ sở.

Đối với thuật toán song song, nhiệm vụ đầu tiên trong PHC là duyệt và phân tách vấn đề thành các nhiệm vụ phụ và thực hiện song song từng nhiệm vụ phụ. Trong trường hợp này, C_t và C_b được biểu diễn dưới dạng: $C_t = \hat{C}_t + \check{C}_t$ và $C_b = \hat{C}_b + \check{C}_b$, trong đó $\hat{C}_t$ và $\hat{C}_b$ là chi phí của phép kiểm tra chồng lấp khối bao và hệ bao tương ứng của BVTT, trong phần đầu tiên tương ứng với duyệt và thu thập các nhiệm vụ phụ. $\check{C}_t$ và $\check{C}_b$ là chi phí của các phép kiểm tra chồng lấp khối lượng và giới hạn khối lượng BVTT trong tất cả các nhiệm vụ phụ.

Chi phí tính toán của thuật toán dựa trên PHC là:

$$T_{Phc} = \hat{C}_t + \hat{C}_b + \left(\check{C}_t + \check{C}_b + C_e\right)/N \tag{2}$$

với N là số nhân.

Bằng cách sử dụng kỹ thuật đề xuất trong phần B nêu trên, bước đầu tiên của phép duyệt BVTT từ nút gốc được loại bỏ. Do tất cả các nút của vùng kiểm tra BVTT được dò theo phương pháp song song, chi phí tính toán của thuật toán dựa trên kỹ thuật đề xuất là:

$$T_{Proposal} = (\lambda * C_b + C_e)/N \tag{3}$$

trong đó λ là tỷ lệ của các phép kiểm tra chồng lấp khối bao giảm, λ là hằng số nhỏ hơn 1 (trong khoảng từ 0,53 đến 0,63 phụ thuộc vào từng bộ dữ liệu cụ thể trong GAMMA).

Từ (2) và (3), có thể thấy rõ sự khác biệt giữa hiệu suất của hai phương pháp khi số lượng nhân tăng lên: Việc tăng tốc thuật toán dựa trên phân cấp song song bị giới hạn bởi $\hat{C}_t$ và $\hat{C}_b$, trong khi thuật toán đề xuất dựa trên phân tách vùng kiểm tra có khả năng mở rộng tốt hơn.

Việc sử dụng khối bao k-DOP là tối ưu cho việc biểu diễn và phát hiện va chạm của các mô hình vật thể biến dạng, nghiên cứu này sử dụng k-DOP làm khối bao cho mô hình vải. Các giá trị của k được đề xuất là 6, 14, 18 và 26 [12,23]. Với k=6 khối bao chính là AABB theo các cạnh song song với các trục tọa độ. Với k=14, các giá trị tọa độ tối thiểu và tối đa của các đỉnh của các tam giác dọc theo 7 trục, được xác định bởi các véctơ (1, 0, 0), (0, 1, 0) , (0, 0, 1), (1, 1, 1), (1, -1, 1), (1, 1, -1) và (1, -1, -1). Khối bao này sử dụng 6 nửa không gian xác định các cạnh như của AABB, cùng với 8 nửa đường chéo bổ sung trong 8 góc của AABB. Với k=18, cũng xuất phát từ 6-DOP (AABB), nhưng tăng thêm 12 nửa đường chéo bổ sung để cắt 12 cạnh của AABB; 12 nửa không gian này được xác định bởi 6 trục, các véctơ chỉ phương (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, -1, 0), (1, 0, -1) và (0, 1, -1). Với k=26, khối bao được xác định bởi sự kết hợp giữa các nửa không gian xác định cho 14-DOP và 18-DOP, sử dụng 6 nửa không gian của AABB, cộng với 8 nửa đường chéo cắt đi các góc, cộng với 12 nửa không gian cắt các cạnh của AABB. Bảng 1 thể hiện thời gian chạy của thuật toán đề xuất (*ms*) với ba bộ dữ liệu mô hình vải của thư viện GAMMA (được mô tả trong mục IV) khi k=6, 14, 18 và 26.

Bảng 1. Bảng so sánh thời gian phát hiện va chạm trung bình (*ms*) của thuật toán đề xuất với các k-DOP khác nhau

Bộ dữ liệu	6-DOP	14-DOP	18-DOP	26-DOP
Princess	15.9	10.3	9.9	11.3
Flamenco	64.7	52.1	48.6	69.7
Cloth-ball	87.4	54	55	57.4

Theo kết quả trong Bảng 1, thuật toán đề xuất chạy nhanh nhất với k=18, do đó khối bao 18-DOP được sử dụng trong phần thử nghiệm tiếp theo.

IV. THỬ NGHIỆM VÀ KẾT QUẢ

A. Thử nghiệm

Ba thuật toán là thuật toán đề xuất (thực hiện song song), thuật toán trong [24] (thực hiện song song) và thuật toán trong [25] (thực hiện tuần tự) được cài đặt bằng ngôn ngữ C++ trong bộ Visual Studio 2010 với OpenMP trên máy tính HP Workstation Z420 GTX750 CPU Intel Xeon Core E5-2665 (8 nhân) 3.1 GHz và RAM 16 GB. BVHs được xây dựng theo phương pháp từ trên xuống (Top-down).

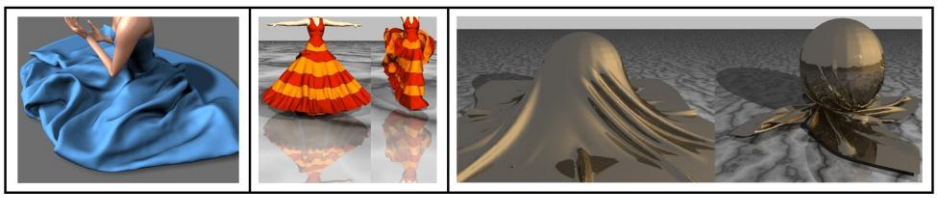
B. Bộ dữ liệu

Ba bộ dữ liệu trong thư viện mở GAMMA (Geometric Algorithms for Modeling, Motion, and Animation) trên website *http://gamma.cs.unc.edu* được sử dụng để đánh giá hiệu suất của các thuật toán, đó là:

Princess (Hình 7a): Mô phỏng một cô gái mặc chiếc váy suông (long flowing skirt) thực hiện động tác ngồi xuống và đứng lên nhẹ nhàng. Mô hình chiếc váy gồm 40000 tam giác.

Flamenco (Hình 7b): Mô phỏng một cô gái mặc chiếc váy xòe có nhiều lớp diềm vải trang trí (skirt with ruffles) thực hiện các động tác múa theo vũ điệu Flamenco. Bộ dữ liệu này có số lượng vải tự va chạm rất lớn. Mô hình chiếc váy gồm 49000 tam giác.

Cloth-ball (Hình 7c): Mô phỏng một mảnh vải (gồm 92230 tam giác) phủ trên đỉnh của quả cầu (gồm 760 tam giác) và khi quả cầu cuộn tròn cuốn theo mảnh vải dẫn đến có nhiều phần vải tự va chạm.

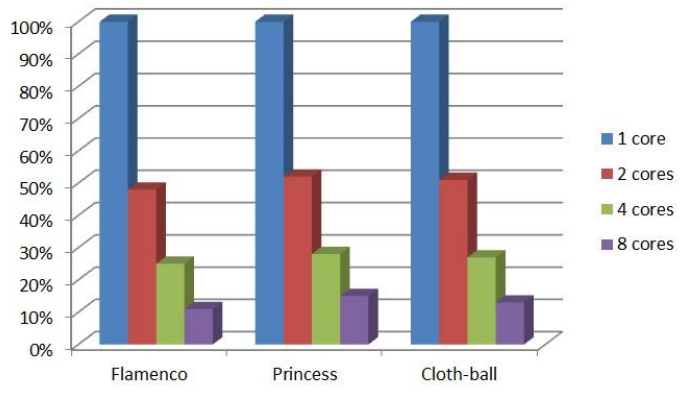


Hình 4. Minh họa trên các bộ dữ liệu: (a) Bộ dữ liệu Princess, (b) Bộ dữ liệu Flamenco, (c) Bộ dữ liệu Cloth-ball

Các bộ dữ liệu có nhiều frames, thuật toán thực hiện phát hiện va chạm và tính toán thời điểm tiếp xúc nếu có va chạm xảy ra.

C. Kết quả

Khi 8 nhân CPU được sử dụng, tốc độ tăng tốc của thuật toán đề xuất thử nghiệm với bộ dữ liệu Princess là 6.4 lần, với bộ dữ liệu Flamenco là 7.7 lần và với bộ dữ liệu Cloth-ball là 6.68 lần.



Hình 5. Kết quả chạy thuật toán dựa trên phân tách vùng kiểm tra đối với các bộ dữ liệu cụ thể

V. PHÂN TÍCH VÀ ĐÁNH GIÁ

A. So sánh

Từ kết quả trong phần C mục IV cho thấy thuật toán đề xuất đã tăng tốc 6,4 - 7,7 lần so với thuật toán tuần tự trong [25], còn thuật toán dựa trên phân cấp song song trong [24] chỉ tăng tốc được 3,9 - 5,8 lần so với thuật toán tuần tự trong [25]. Bảng 2 cho thấy thời gian phát hiện va chạm trung bình (*ms*) của ba thuật toán: Thuật toán đề xuất (thực hiện song song), thuật toán trong [24] (thực hiện song song) và thuật toán trong [25] (thực hiện tuần tự).

Bảng 2. Bảng so sánh thời gian phát hiện va chạm trung bình (*ms*) của các thuật toán

Bộ dữ liệu	Thuật toán đề xuất	Thuật toán trong [24]	Thuật toán trong [25]
Princess	9.9	15.9	63.5
Flamenco	48.6	63.5	374.4
Cloth-ball	55	81.7	367.5

Hầu hết các thuật toán trước đây [8, 12, 14, 17, 31] được công bố để xử lý các mô hình vật thể rắn và bài báo này phát triển kỹ thuật phát hiện tự va chạm đối với mô hình vải. Kỹ thuật đề xuất đạt được sự cải thiện hiệu suất đáng kể so với các thuật toán trước được thiết kế cho mô phỏng vải hoặc va chạm vải [26, 27]. Kỹ thuật đề xuất có hai ưu điểm nổi bật: Thứ nhất là thời gian chạy của việc kiểm tra chồng lấp khối bao và duyệt BVH được giảm đáng kể đối với các mô hình vải; thứ hai là chiến lược phân tách nhiệm vụ chi tiết tốt để song song hóa, rất hữu ích cho việc thực thi song song trên các bộ xử lý đa nhân.

B. Hạn chế

Thứ nhất, tốc độ tăng của kỹ thuật đề xuất tăng tuyến tính khi tăng số nhân CPU; khi số lượng nhân tăng lên thì chi phí thực hiện song song cũng tăng. Thứ hai, chi phí sử dụng bộ nhớ là ở mức cao do phải duy trì vùng kiểm tra BVTT; khi số lượng các cặp va chạm tăng lên thì kích thước của vùng kiểm tra được tăng lên đáng kể.

VI. KẾT LUẬN VÀ ĐỊNH HƯỚNG NGHIÊN CỨU TIẾP THEO

Trong bài báo này, chúng tôi trình bày một thuật toán phát hiện va chạm song song cho các mô hình vải dựa trên kỹ thuật tái cấu trúc thích nghi, kết hợp với tập rút gọn để giảm số lượng phép kiểm tra cơ sở. Thay vì xây dựng cấu trúc cây nhị phân để biểu diễn BVTT như các phương pháp truyền thống, kỹ thuật đề xuất xây dựng cấu trúc cây tam phân cho phép sử dụng bộ nhớ hiệu quả hơn. Trong thử nghiệm, kỹ thuật phân tách vùng kiểm tra cải thiện đáng kể hiệu năng của thuật toán phát hiện va chạm trên hệ thống CPU đa nhân. Kỹ thuật đề xuất cho phép tăng nhanh tốc độ phát hiện va chạm gấp 6,4; 6,68 và 7,7 lần với ba bộ dữ liệu mô hình vải trong thư viện mở GAMMA so với kỹ thuật ICCD [25]. Trong tương lai, chúng tôi dự kiến mở rộng cách tiếp cận này với GPU.

VII. LỜI CẢM ƠN

Để hoàn thành bài báo này, chúng tôi xin chân thành cảm ơn nhóm nghiên cứu GAMMA, Tang và cộng sự [24, 25] vì đã chia sẻ mã nguồn mở và các bộ dữ liệu thử nghiệm.

TÀI LIỆU THAM KHẢO

[1] U. Assarsson, P. Stenström, "A case study of load distribution in parallel view frustum culling and collision detection", In Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, pp. 663-673, 2001.

[2] G. Bergen, "Efficient collision detection of complex deformable models using AABB trees", Journal of Graphics Tools, vol. 2, no. 4, pp. 1-14, 1997.

[3] G. Bradshaw, C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction", ACM Transaction on Graphics, vol. 23, no. 1, pp. 1-26, 2004.

[4] W. Chao, et al., "Research on Bounding Volume Boxes Collision Detection Algorithm in Virtual Reality Technology", In Proceedings of the International Conference of Sensor Network and Computer Engineering (ICSNCE 2018), pp. 84-88, 2018.

[5] G. Chaoyang, "An Improved Algorithm of the Collision Detection Based on OBB", In Proceedings of the International Conference of Sensor Network and Computer Engineering (ICSNCE 2018), pp. 44-47, 2018.

[6] Y. K. Chen, et al., "High-performance physical simulations on next-generation architecture with many cores", Intel Technology Journal, vol. 11, no. 3, pp. 251-261, 2007.

[7] F. M. Chitalu, et al., "Bulk-synchronous Parallel Simultaneous BVH Traversal for Collision Detection on GPUs", In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 4-9, 2018.

[8] S. A. Ehmann, M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition", Journal of Computer Graphics Forum, vol. 20, pp. 500-510, 2001.

[9] S. Gottschalk, et al., "Obbtree: A hierarchical structure for rapid interference detection", In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, USA, pp. 171-180, 1996.

[10] I. Grinberg, Y. Wiseman, "Scalable parallel collision detection simulation", In Proceedings of the Signal and Image Processing, pp. 380-385, 2007.

[11] D. Kim, et al., "PCCD: Parallel Continuous Collision Detection", In Proceedings of the SIGGRAPH '09, pp. 50, 2009.

[12] J. Klosowski, et al., "Efficient collision detection using bounding volume hierarchies of K-DOPs", IEEE Transaction on Visualization and Computer Graphics, vol. 4, no. 1, pp. 21-37, 1998.

[13] V. Kumar, A. Y. Grama, "Scalable load balancing techniques for parallel computers", Journal of Parallel and Distributed Computing, vol. 22, pp. 60-79, 1994.

[14] E. Larsen, et al., "Fast Distance Queries with Rectangular Swept Sphere Volumes", In Proceedings of the ICRA IEEE International Conference on Robotics and Automation, pp. 3719-3726, 2000.

[15] T. Larsson, T. Akenine-Möller, "A dynamic bounding volume hierarchy for generalized collision detection", Journal of Computers and Graphics, vol. 30, no. 3, pp. 451-460, 2006.

[16] D. Meister, J. Bittner, "Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction", IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 3, pp. 1345-1353, 2018.

[17] M. A. Otaduy, M. C. Lin, "CLODs: Dual hierarchies for multiresolution collision detection", In Proceeding of the Eurographics Symposium on Geometry, pp. 94-101, 2003.

[18] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garment", In Proceedings of the Graphics Interface, pp. 177-189, 1997.

[19] V. N. Rao, V. Kumar, "Parallel depth-first search. Part I. Implementation", International Journal of Parallel Programming, vol. 16, no. 6, pp. 479-499, 1987.

[20] S. Redon, et al., "Fast continuous collision detection between rigid bodies", Journal of Computer Graphics Forum, vol. 21, no. 3, pp. 279-288, 2002.

[21] A. Reinefeld, V. Schnecke, "Work-load balancing in highly parallel depth-first search", In Proceedings of the Scalable High Performance Computing Conference, pp. 773-780, 1994.

[22] A. Sud, et al., "Fast proximity computation among deformable models using discrete voronoi diagrams", In Proceedings of ACM SIGGRAPH, pp. 1144-1153, 2006.

[23] M. Tang, et al., "PSCC: Parallel Self-Collision Culling with Spatial Hashing on GPUs", In Proceedings of the ACM on Computer Graphics and Interactive Techniques, pp.18-29, 2018.

[24] M. Tang, et al., "MCCD: Multi-core collision detection between deformable models using front-based decomposition", Graphical Models, vol. 72, no. 2, pp, 7-23, 2010.

[25] M. Tang, et al., "ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling", IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 4, pp. 544-557, 2009.

[26] B. Thomaszewski, et al., "Parallel techniques for physically based simulation on multi-core processor architectures", Journal of Computers and Graphics, vol. 32, no. 1, pp. 25-40, 2008.

[27] B. Thomaszewski, W. Blochinger, "Physically based simulation of cloth on distributed memory architectures", Journal of Parallel Computing, vol. 33, no. 6, pp. 377-390, 2007.

[28] I. Wald, et al., "State of the art in ray tracing animated scenes", In Proceedings of the Eurographics Association, pp. 89-116, 2007.

[29] X. Wang, et al., "Efficient BVH-based Collision Detection Scheme with Ordering and Restructuring", Journal of Computer Graphics Forum, vol. 37, no. 2, pp. 227-237, 2018.

[30] T. Wang, et al., "Accurate self-collision detection using enhanced dual-cone method", Computers & Graphics, vol. 73, pp. 70-79, 2018.

[31] R. Weller, G. Zachmann, "Kinetic separation lists for continuous collision detection of deformable objects", In Proceedings of the Third Workshop in Virtual Reality Interactions and Physical Simulation (Vriphys), Madrid, Spain, pp. 33-42, 2006.

[32] S. E. Yoon, D. Manocha, "Cache-efficient layouts of bounding volume hierarchies", Journal of Computer Graphics Forum, vol. 25, no. 3, pp. 507-516, 2006.

[33] G. Zachmann, R. Weller, "Kinetic bounding volume hierarchies for deforming objects", In Proceedings of the ACM Conference on Virtual Reality Continuum and its Applications, pp. 189-196, 2006.

[34] X. Zhang, et al., "Continuous collision detection for articulated models using taylor models and temporal culling", ACM Transactions on Graphics, vol. 26, no. 3, pp. 10-15, 2007.

A FAST COLLISION DETECTION TECHNIQUE FOR VIRTUAL CLOTH USING PARALLEL COMPUTING METHOD

Nghiem Van Hung, Dang Van Duc, Trinh Hien Anh, Hoang Viet Long, Nguyen Van Can

ABSTRACT: *Detecting the collision of cloth in a 3D environment is currently one of the most computationally demanding tasks in virtual reality systems. The natural interaction of cloth with objects and themselves often fold, roll, … make detecting the collision and self-collision of cloth computations more challenging. By using tree structures to accelerate collision queries, we present an parallel algorithm for fast collision detection between cloth models using multi-core processors. In this paper, our approach distributes the computation among multiple cores by using front-based decomposition and efficient techniques to reduce the number of computations. Test results showed efficacy with 03 cloth models benchmarks in GAMMA on PC using Intel Xeon Core E5-2665 CPU (8 cores) with C++ language in Visual Studio 2010 (OpenMP) and observe up to 6.4, 6.68 and 7.7 speedups respectively.*

Keywords: *Collision detection, self-collision, cloth model, parallel computing.*