ĐẠI HỌC QUỐC GIA TP. HCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

Tên đề tài tiếng Việt:

Mô phỏng Trạng thái Game Billiards

Tên đề tài tiếng Anh:

Simulation for Billiards Game State

Khoa: Khoa học Máy tính.

Lớp:

CS231.L21.KHTN - Nhập môn Thị giác Máy tính

CS105.L21.KHTN - Đồ họa Máy tính

Giảng viên hướng dẫn: TS. Mai Tiến Dũng

Tham gia thực hiện:

STT	Họ và tên	MSSV	Vai trò
1	Hồ Chung Đức Khánh	19520624	Trưởng nhóm
2	Nguyễn Thị Minh Phương	19522065	Thành viên
3	Võ Minh Hiếu	19520084	Thành viên

MỤC LỤC

Chương 1:
TỔNG QUAN

1. Giới thiệu bài toán

Mô phỏng 3D là hình thức tái hiện lại trạng thái hay hoạt động của các vật thể, sự kiện trong thực tế trở thành mô hình 3D trong máy tính. Mô phỏng được sử dụng để kiểm thử các hoạt động không thể thực hiện, hoặc nguy hiểm, hoặc không được chấp nhận trong hệ thống thực.

Áp dụng khái niệm mô phỏng trên, bài toán của chúng tôi nhận vào hình chụp trạng thái đang diễn ra trên bàn bi-da và trả về trạng thái mô phỏng 3D trong game tương ứng.

Đồ án "**Mô phỏng Trạng thái game Billiards**" là một sản phẩm kết hợp giữa hai lĩnh vực: Thị giác Máy tính và Đồ họa Máy tính. Game của chúng tôi mô phỏng được vị trí của banh trong ảnh đầu vào, có đầy đủ các hiện tượng vật lý như trong thực tế và các tính năng thông thường như những game bi-da khác. Chúng tôi tạo ra sản phẩm này vì muốn phục vụ người chơi bi-da có thể tìm ra nước đi chính xác thông qua việc chụp hình trạng thái bên ngoài và đưa vào game mô phỏng, sau đó người chơi có thể dễ dàng thử nghiệm các nước đi thông qua game.

2. Cấu trúc

Báo cáo này là một phần của đồ án cuối kỳ môn Nhập môn Thị giác Máy tính (CS231.L21.KHTN) và Đồ họa Máy tính (CS105.L21.KHTN), do thầy Mai Tiến Dũng giảng dạy và hướng dẫn. Thông qua đồ án lần này, nhóm chúng tôi mong rằng sẽ thực hiện được những nội dung sau:

- Tìm hiểu về các hàm của thư viện OpenCV và có thể áp dụng để giải quyết những bài toán cụ thể.
- Tìm hiểu về thư viện OpenGL để có thể render ra được game.

- Tìm hiểu về công cụ Blender để tạo ra những vật thể 3D phức tạp.
- Thử nhiều phương pháp để giải quyết một bài toán cụ thể và so sánh, đánh giá để chọn ra được phương pháp phù hợp với từng bài toán.
- Tạo ra được một sản phẩm có thể kết hợp, liên kết được kiến thức của hai mảng Thị giác Máy tính và Đồ họa Máy tính.
- Rút ra được ưu điểm, hạn chế và hướng phát triển của sản phẩm.
- Tổng hợp các kiến thức, các bước thực hiện đồ án cụ thể vào một bài báo cáo hoàn chỉnh.

Cấu trúc của bài báo cáo chúng tôi gồm **năm** phần:

Chương 1: Tổng quan

Giới thiệu tổng quan về bài toán lớn, cấu trúc của nó cùng với phân công nhiệm vụ trong nhóm.

Chương 2: Các bài toán và phương pháp liên quan

Tìm hiểu sâu hơn về các bài toán con và những phương pháp để thực hiện các bài toán đó.

Những bài toán con bao gồm: định vị góc bàn, phát hiện banh bi-da, biến đổi tọa độ sang không gian 3D và mô phỏng game.

Chương 3: Kết quả và thảo luận

Dựa trên bộ dữ liệu thực tế để có thể so sánh, đánh giá các phương pháp của những bài toán con của chương 2, từ đó chọn ra phương pháp phù hợp nhất.

Chương 4: Xây dựng ứng dụng minh họa

Giới thiệu mô hình máy chủ - máy khách để xây dựng game, hướng dẫn người dùng cài đặt và sử dụng chương trình.

Chương 5: Ưu điểm, hạn chế và hướng phát triển

Kết luận ưu điểm, hạn chế, hướng phát triển của đồ án lần này và so sánh với những đồ án đã có sẵn.

3. Bảng phân công nhiệm vụ

Họ và tên	Phân công nhiệm vụ	Mức độ hoàn thành
Hồ Chung Đức Khánh	- Thực hiện bài toán Định vị góc bàn. - Xây dựng mô hình client-server cho game. - Đóng gói sản phẩm dưới dạng Docker cho người dùng. - Làm slide và viết báo cáo đồ án.	100%
Nguyễn Thị Minh Phương	- Khảo sát và đánh giá các dự án liên quan. - Tìm hiểu về Blender và tạo vật thể 3D. - Thực hiện bài toán phát hiện banh bi-da và biến đổi tọa độ. - Làm slide và viết báo cáo đồ án.	100%
Võ Minh Hiếu	- Tìm hiểu về OpenGL. - Thực hiện bước mô phỏng render. - Tìm hiểu và mô phỏng vật lí cho banh bi-da. - Làm slide và viết báo cáo đồ án.	100%

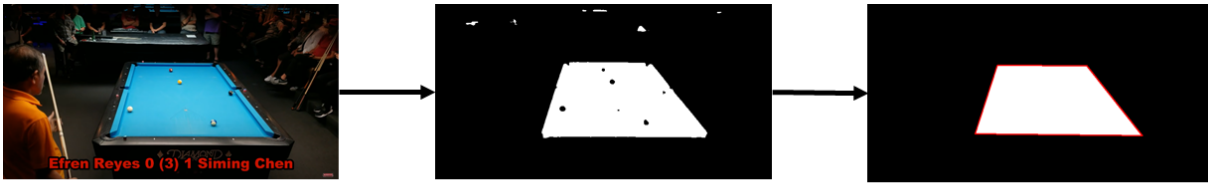
Chương 2:
CÁC BÀI TOÁN VÀ PHƯƠNG PHÁP LIÊN QUAN

1. Định vị góc bàn

1.1. Định nghĩa bài toán

Nhằm tăng mức độ hiệu quả của bước phát hiện vật thể banh bi-da, cũng như giúp xác định chính xác vị trí của mặt bàn bi-da trong ảnh trước khi ánh xạ chúng vào môi trường mô phỏng, ở phần này nhóm chúng tôi đề xuất bài toán định vị góc bàn, đồng thời khảo sát và tìm hiểu một số phương pháp liên quan để tiếp cận với bài toán đưa ra.

Chúng tôi định nghĩa bài toán định vị góc bàn với đầu vào là một bức ảnh có chứa mặt bàn bi-da, đầu ra là tọa độ bốn điểm tương ứng với bốn góc của bàn.



a) b) c)

Hình 1: Các bước thực hiện để xác định bốn góc bàn.
a) Ảnh đầu vào. b) Sau khi phân đoạn. c) Sau khi ước lượng tứ giác.

Thông thường, ảnh đầu vào sẽ được chụp bởi các thiết bị di động dưới các điều kiện chiếu sáng khác nhau, dẫn đến việc màu sắc của mặt bàn bi-da trong ảnh trở nên đa dạng. Một điểm khó khăn khác của bài toán này là bốn góc của bàn bi-da không nhất thiết phải cùng xuất hiện trong ảnh đầu vào, trong trường hợp này các phương pháp được đề xuất cần dự đoán những tọa độ nằm bên ngoài diện tích bức ảnh. Để đưa ra được một giải pháp hiệu quả và ổn định đối với những thách thức kể trên, chúng tôi chia bài toán thành hai bước chính: Phân đoạn ảnh và Ước lượng tứ giác. (Hình 1)

1.2. Phân đoạn ảnh

Một trong những hướng tiếp cận cơ bản cho bài toán phát hiện vật thể là phân đoạn ảnh dựa trên màu sắc. Quan sát cho thấy bàn bi-da thường xuất hiện trong ảnh có màu xanh lam hoặc xanh lục với đa dạng các tính chất như đậm/nhạt hoặc sáng/tối, việc phân đoạn ảnh dựa trên không gian màu HSV (Hue Saturation Value) được cho là hiệu quả hơn so với không gian màu RGB (Red Green Blue) vì khoảng cách giữa các gam màu đậm/nhạt hoặc sáng/tối sẽ nhỏ hơn khi so sánh trong không gian HSV.

Các thuật toán phân đoạn ảnh đơn giản hoạt động bằng cách giới hạn dải màu xanh lam hoặc xanh lục dựa trên một khoảng giá trị được định nghĩa trước của ba kênh màu Hue, Saturation và Value. Cụ thể, gọi giá trị màu sắc của điểm ảnh thứ i trong ảnh là $p_i = (h_i, s_i, v_i)$, chúng tôi định nghĩa thêm 6 tham số $h_a, h_b, s_a, s_b, v_a, v_b$ tương ứng là giá trị thấp nhất và cao nhất của mỗi kênh màu, thuật toán phân đoạn sẽ tạo ra một bức ảnh nhị phân mới với công thức:

$$p'_i = 1 \Leftrightarrow \begin{cases} h_a \leq h_i \leq h_b \\ s_a \leq s_i \leq s_b \\ v_a \leq v_i \leq v_b \end{cases}$$

Đối với trường hợp bàn bi-da màu xanh lam, chúng tôi thiết lập giá trị $h_a = 80$, $h_b = 130$. Tương tự, các giá trị $h_a = 40, h_b = 70$ được áp dụng cho trường hợp bàn bi-da màu xanh lục. Cả hai loại màu đều dùng chung các tham số $s_a = 60, s_b = 255, v_a = 60, v_b = 255$.

1.2.1. Ngưỡng cố định

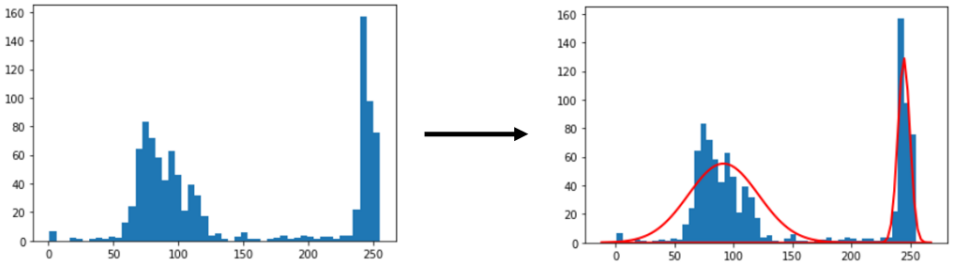
Dễ nhận thấy việc chọn các tham số cho thuật toán hoàn toàn phụ thuộc vào thực nghiệm và cần sự tinh chỉnh để có thể tìm ra bộ tham số hoạt động tốt trong đa số các trường hợp. Trên thực tế, gần như không thể tìm được một bộ tham số hoàn hảo vì ngưỡng được chọn có thể hoạt động tốt trên trường hợp này tuy nhiên lại quá cao hoặc quá thấp khi áp dụng lên trường hợp khác. Ngoài ra, việc phân đoạn dựa trên một ngưỡng xác định trước cũng không hiệu quả trong những trường hợp mặt bàn bi-da có màu tương đồng với khung cảnh xung quanh (Hình 2). Từ các quan sát trên có thể thấy việc cố định ngưỡng cho bài toán phân đoạn còn tồn đọng nhiều hạn chế và có thể được cải thiện bằng những phương pháp xác định ngưỡng linh hoạt.

Hình 2: Mặt bàn có màu tương đồng với màu nền.

1.2.2. Ngưỡng linh hoạt

Nhận thấy trong 6 tham số $h_a, h_b, s_a, s_b, v_a, v_b$, chỉ tham số Hue và Value cần được xác định trước vì bài toán cần định nghĩa rõ khoảng màu của xanh lam và xanh lục cũng như mức độ sáng tối thường tương đồng nhau trên toàn bộ ảnh. Mặt khác thì kênh Saturation thể hiện cho mức độ đậm/nhạt của màu sắc, và giá trị sẽ thay đổi tùy thuộc vào điều kiện ánh sáng hoặc chất lượng của ảnh được chụp, vì vậy các tham số Saturation phải được xác định một cách linh hoạt thay vì chọn một ngưỡng cố định.

Để tìm được ngưỡng phù hợp cho kênh Saturation của mỗi ảnh, chúng tôi áp dụng phương pháp thống kê histogram và ước lượng phân phối màu của mặt bàn dựa trên Mô hình Hỗn hợp Gaussian (Gaussian Mixture Model). (Hình 3)



Hình 3: Sử dụng Mô hình Hỗn hợp Gaussian để ước lượng phân phối màu.

Cụ thể, đầu tiên bức ảnh đầu vào sẽ được phân đoạn dựa trên ngưỡng Hue và Value để chọn ra những điểm ảnh có giá trị màu cần quan tâm. Kế tiếp, thực hiện đếm và thống kê trên tập hợp những điểm ảnh được phân đoạn để thu được histogram cho màu xanh. Giả sử màu sắc trong bức ảnh xuất hiện tạo thành tập hợp nhiều phân phối chuẩn, có thể sử dụng Mô hình Hỗn hợp Gaussian để ước lượng các phân phối này.

Ở đây, chúng tôi ước lượng $k = 2$ phân phối chuẩn và chọn phân phối ngoài cùng bên phải của histogram vì bàn bi-da thường có màu xanh đậm và nổi bật hơn màu nền.

1.3. Ước lượng tứ giác

Ước lượng tứ giác có mức độ ảnh hưởng lớn trong quy trình giải quyết bài toán định vị góc bàn vì bước này đóng vai trò như cầu nối giữa kết quả phân đoạn hình ảnh và đầu ra cuối cùng của bài toán (tọa độ bốn góc của bàn bi-da).

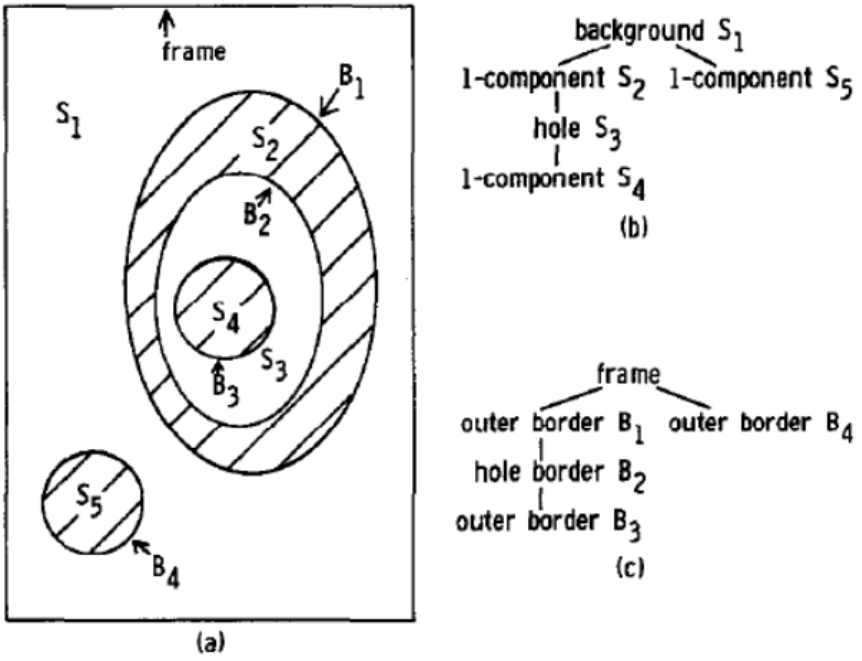
Để tiếp cận bài toán ước lượng tứ giác, đầu tiên chúng tôi thực hiện ước lượng đa giác dựa trên kết quả phân đoạn ảnh thông qua hai thuật toán Contour và Bao lồi (Convex hull). Vì đầu ra của hai thuật toán này là các đa giác với rất nhiều đỉnh, trong khi kết quả cần tìm chỉ là một tứ giác, chúng tôi áp dụng thuật toán Douglas-Peucker để đơn giản hóa và giảm bớt số đỉnh của đa giác tìm được. Với đa giác đã được giản lược hóa, thực hiện duyệt vét cạn để tìm ra cặp cạnh song song dài nhất để lập tứ giác cần tìm, từ đó tiếp tục vét cạn để xác định hai cạnh còn lại của tứ giác.

Sau khi tìm được bốn cạnh của tứ giác bằng phương pháp trên, cần xác định các đỉnh của tứ giác theo thứ tự kim đồng hồ để có thể đưa vào bước ánh xạ sang không gian 3D. Dựa trên ý tưởng của thuật toán Graham, chúng tôi tính trọng tâm của tứ giác và thực hiện sắp xếp bốn đỉnh theo chiều kim đồng hồ của góc tạo bởi vector đơn vị và vector nối từ trọng tâm đến mỗi đỉnh. Kết quả thu được là tập hợp bốn đỉnh được sắp xếp theo chiều kim đồng hồ.

1.3.1. Contour

Có nhiều thuật toán để phát hiện contour trong ảnh như thuật toán Suzuki, thuật toán Moore-neighbor, thuật toán radial sweep, v.v. Trong phạm vi bài báo cáo này, chúng tôi áp dụng thuật toán Suzuki [2] để tìm contour của bàn bi-da từ kết quả phân vùng.

Ý tưởng của thuật toán Suzuki dựa trên việc phân tầng mối quan hệ giữa các đường biên xuất hiện trong ảnh, đồng thời phân biệt đường biên thành hai loại: biên trong và biên ngoài. Theo đó, thuật toán giả sử rằng các vật thể có độ dày khác không, vì vậy sẽ có lên đến hai đường biên nằm tương ứng ở mặt ngoài và mặt trong của vật thể. Ở hình x có thể thấy vật thể S2 có biên ngoài B1 và biên trong B2, tuy nhiên vật thể S5 chỉ có duy nhất biên ngoài B4.



Hình 4: Kết quả nhận diện của thuật toán Suzuki.
Nguồn: Suzuki et al (1985).

Nhờ vào việc phân biệt giữa biên trong và biên ngoài, thuật toán Suzuki có thể tìm ra được các lỗ (hole) bên trong vật thể, và có thể suy diễn được mối quan hệ phân tầng giữa các vật thể hoặc đường biên với nhau. Chẳng hạn, trong hình 4b) có thể thấy vật thể S2 chứa lỗ S3 và lỗ S3 chứa vật thể S4, hoặc trong hình 4c) thể hiện được mối quan hệ biên ngoài B1 chứa biên trong B2 và biên trong B2 chứa biên ngoài B3.

Áp dụng đối với bài toán ước lượng đa giác, thuật toán Suzuki sẽ tìm một đường biên phù hợp để xấp xỉ mặt bàn bi-da sau khi được phân vùng. Thuật toán được cài đặt trong thư viện OpenCV với hàm `cv2.findContours()`.

1.3.2. Bao lồi

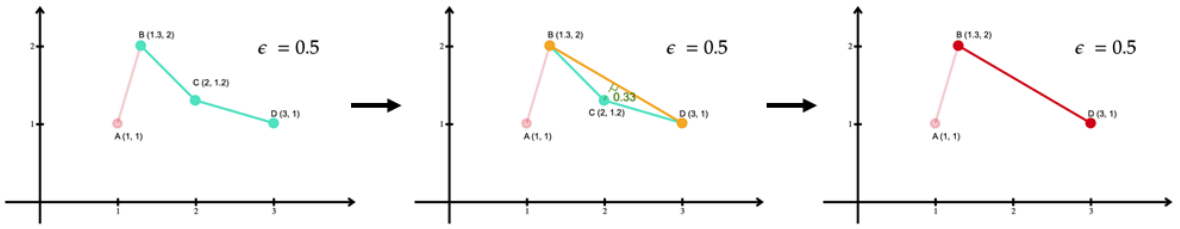
Đối với bài toán tìm bao lồi cũng có nhiều cách tiếp cận, tiêu biểu như thuật toán gói quà với độ phức tạp tính toán $O(n^2)$, hay thuật toán Graham với $O(n \log n)$. Ở đây chúng tôi sử dụng thuật toán Sklansky với độ phức tạp tuyến tính nhằm đảm bảo thời gian tính toán nằm ở mức chấp nhận được vì n tỉ lệ thuận với kích thước ảnh đầu vào.

Thuật toán Sklansky đã được cài đặt sẵn trong thư viện OpenCV thông qua hàm `cv2.convexHull()`.

1.3.3. Thuật toán Douglas-Peucker

Thông thường, kết quả nhận được từ hai thuật toán Suzuki và Sklansky là các đa giác phức tạp với số đỉnh có thể lên đến hàng trăm hoặc thậm chí hàng ngàn. Tuy nhiên bài toán xác định tứ giác chỉ quan tâm đến các đa giác có bốn đỉnh, cần có một phương pháp để tối thiểu hóa số đỉnh của đa giác mà vẫn giữ được mức độ tương đồng so với hình dạng thực tế.

Cách hoạt động của thuật toán Douglas-Peucker (DP) [3] dựa trên ý tưởng đệ quy. Xem đa giác như tập hợp các điểm $p_1, p_2, \ldots, p_n$, tìm điểm p_i có khoảng cách lớn nhất đến đường thẳng nối p_1 và p_n, Nếu khoảng cách này nhỏ hơn ε cho trước, tập hợp các điểm $p_2, p_3, \ldots, p_{n-1}$ có thể được lược bỏ. Ngược lại nếu khoảng cách lớn hơn ε, gọi đệ quy thuật toán DP trên tập hợp $p_1, p_2, \ldots, p_i$ và $p_i, p_{i+1}, \ldots, p_n$.



Hình 5: Thuật toán Douglas-Peucker trong trường hợp khoảng cách nhỏ hơn ε. Nguồn: medium.com.

Kết quả mà thuật toán DP trả về sẽ là một đa giác mới có số đỉnh ít hơn, sao cho sai số khoảng cách giữa các điểm trên đa giác cũ và đa giác nhận được không vượt quá ε. Thuật toán DP đã được cài đặt sẵn trong thư viện OpenCV thông qua hàm `cv2.approxPolyDP()`.

2. Phát hiện banh bi-da

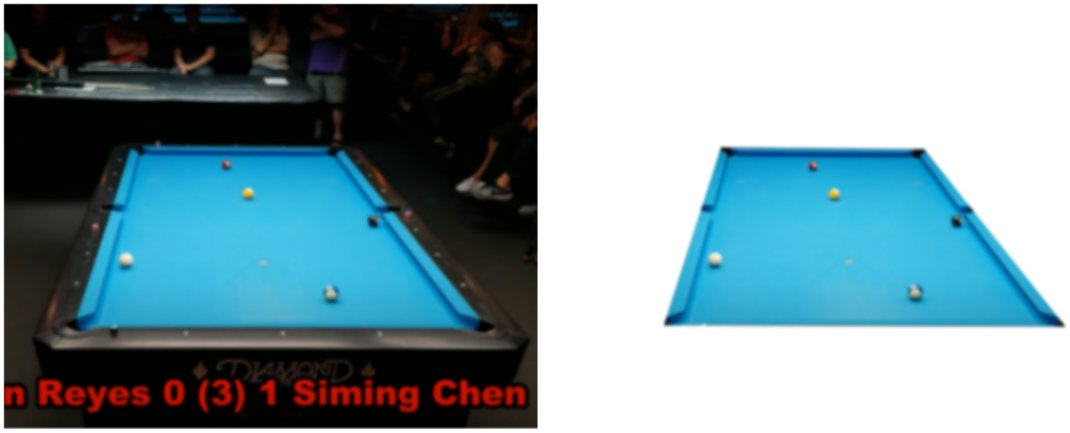
2.1. Định nghĩa bài toán

Để có thể có được vị trí của banh bi-da trong game, đầu tiên cần phải xác định được vị trí của banh trong hình chụp 2D. Bài toán này nhận đầu vào là hình chụp bàn bi-da cùng với tọa độ của 4 góc bàn, sau đó trả về tọa độ của các quả banh bi-da có trong hình.

2.2. Tiền xử lý hình ảnh

Chúng tôi sử dụng hình ảnh gốc kết hợp với tọa độ của 4 góc bàn được trả về từ bài toán định vị góc bàn để xóa đi phông nền xung quanh bàn. Việc chỉ giữ lại

bàn bida giúp cho các phương pháp phát hiện banh bida không phát hiện sai những vật thể hình tròn khác nằm xung quanh nền.



Hình 6: Giai đoạn tiền xử lý hình ảnh
a) Trước khi tách nền. b) Sau khi tách nền.

2.3. Các phương pháp liên quan

Để phát hiện được các quả banh bida trong hình, ta cần sử dụng những hàm giúp ta phát hiện những vật thể hình tròn.

Chúng tôi đã thử qua 3 phương pháp để phát hiện banh bida:

1. Sử dụng kết hợp Canny edge detection và Contour.
2. Sử dụng kết hợp Canny edge detection và Hough circle transform.
3. Sử dụng hàm Hough circle transform.

Chúng tôi đã kiểm thử trên hình ảnh đơn giản chỉ gồm 3 banh trước, nếu phương pháp nào khả thi thì áp dụng lên hình ảnh bàn bida.

2.3.1. Canny edge detection + Contour

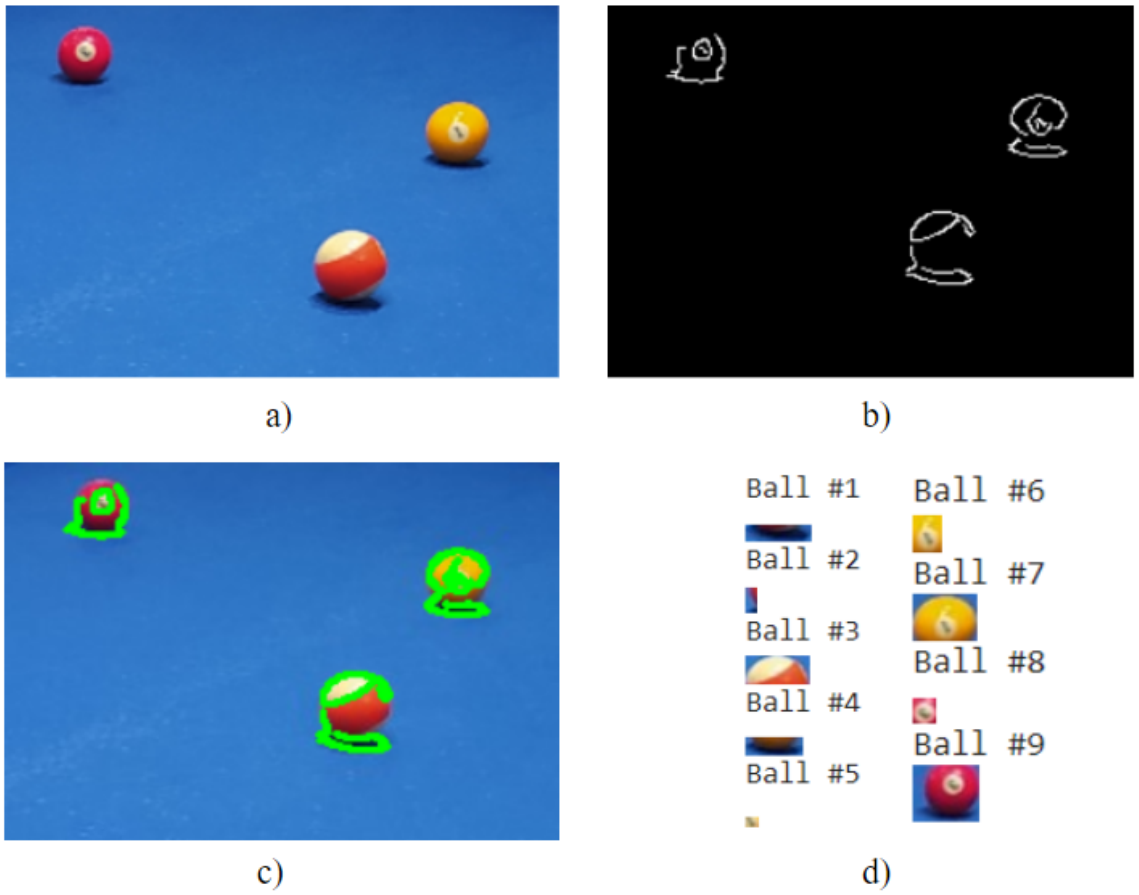
Đây là phương pháp sử dụng Canny edge detection để phát hiện cạnh của banh, sau đó kết hợp với Contour để làm nổi bật viền của các cạnh đã phát hiện.

- Canny edge detection: *cv2.Canny()*.

  Hình ảnh truyền vào phải được chuyển đổi sang gray trước *(cv2.cvtColor)*, sau đó làm mờ bằng *cv2.GaussianBlur()* để bỏ bớt các cạnh không quan trọng.

- Contour: *cv2.findContour()*.

  Tham số truyền vào là hình ảnh kết quả trả về của hàm Canny.



Hình 7: Kết hợp Canny Edge Detection và Contour

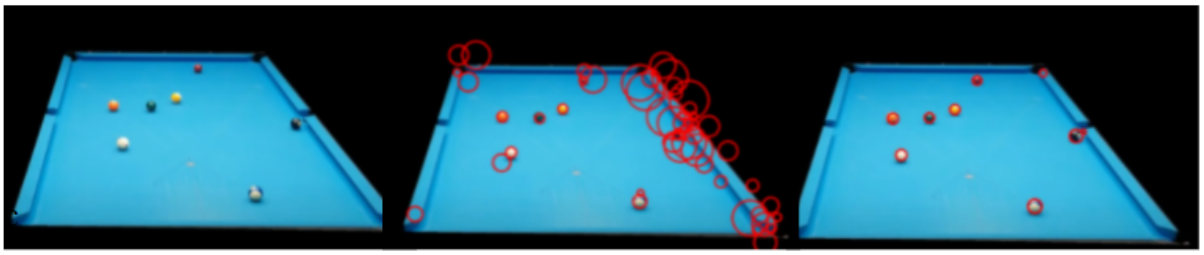
a) Hình gốc. b) Canny edge detection. c) Contour. d) Kết quả.

2.3.2. Hough circle transform

OpenCV có cung cấp sẵn hàm để phát hiện những vật thể hình tròn là Hough circle transform *(cv2.HoughCircles())*

Tuy nhiên, để hàm Hough phát huy được tối đa hiệu quả của nó, cần phải điều chỉnh bộ tham số phù hợp. Có 5 tham số quan trọng cần phải quan tâm đó là: minDist, param1, param2, minRadius, maxRadius. Cụ thể minDist là khoảng cách tối thiểu của các quả banh được phát hiện, maxRadius/minRadius là bán kính lớn nhất và nhỏ nhất có thể của banh, param1 là ngưỡng threshold cao để dò cạnh bằng Canny, param2 là accumulation threshold cho tâm vòng tròn. Param2 càng nhỏ sẽ phát hiện càng nhiều hình tròn, khả năng sai cao, param2 càng lớn thì khả năng sai giảm tuy nhiên số quả banh không được phát hiện cũng tăng theo, nên cần phải cân nhắc và thử để lựa chọn tham số phù hợp.

Kết quả trả về sẽ là list các *(x,y,r)*, trong đó *(x,y)* là tọa độ của quả banh trong hình và *r* là bán kính.



a) b) c)

Hình 8: Sử dụng Hough Circle Transform
a) Hình gốc. b) Trước khi điều chỉnh tham số. c) Sau khi điều chỉnh tham số.

2.3.3. Canny edge detection + Hough circle transform

Phương pháp thử cuối cùng là sử dụng Canny edge detection (đã được trình bày ở 2.3.1) để phát hiện cạnh trước, sau đó kết hợp Hough circle transform (2.3.2) để phát hiện vật thể hình tròn trên những cạnh đã được phát hiện.

a) b) c)

Hình 9: Kết hợp Canny Edge Detection và Hough Circle Transform
a) Hình gốc. b) Canny edge detection. c) Hough circle transform.

3. Biến đổi tọa độ sang không gian 3D

3.1. Định nghĩa bài toán

Sau khi có được tọa độ của banh trong hình chụp, chúng cần phải qua các bước biến đổi để có thể trở thành tọa độ trong không gian 3D có thể sử dụng trong game. Trong đồ án này, chúng tôi sử dụng phép biến đổi *perspective transform*. Bài toán nhận tham số đầu vào gồm tọa độ 4 góc bàn và tọa độ của banh trong hình chụp, và cho đầu ra là tọa độ của banh trong không gian 3D, cụ thể là góc chính diện nhìn từ trên xuống của bàn bi-da.

Bên cạnh đó, nhờ vào bước biến đổi này chúng tôi cũng đã phát hiện thêm phương pháp loại bỏ banh sai và tăng được độ chính xác.

3.2. Các bước thực hiện

3.2.1. Tìm ma trận biến đổi

Để tìm ma trận biến đổi từ bàn bida chụp ngoài thực tế sang góc chính diện nhìn từ trên xuống, ta sử dụng hàm *cv2.getPerspectiveTransform()*. Tham số truyền vào gồm tọa độ 4 góc bàn thực tế và tọa độ 4 góc bàn của góc chính diện.

Ngoài ra còn có hàm *cv2.warpPerspective()* nhận vào hình chụp và ma trận biến đổi và trả về hình ảnh sau khi biến đổi, giúp ta dễ dàng kiểm tra tính chính xác.

a) b)

Hình 10: Thực hiện phép biến đổi perspective transform
a) Hình gốc. b) Hai trường hợp sau khi biến đổi.

Ta có thể thấy được hình ảnh sau khi biến đổi có thể là 1 trong 2 trường hợp, vì ma trận biến đổi không biết được là cạnh nào là chiều dài, cạnh nào là chiều rộng để có thể kết hợp chính xác. Nên ta sẽ qua bước xác định cạnh chiều dài/chiều rộng của bàn bi-da

3.2.2. Xác định cạnh chiều dài/chiều rộng của bàn bi-da

Vì hình chụp ngoài không gian thực tế không còn giữ đúng tỉ lệ độ dài, nên ta không thể nói "cạnh chiều dài là cạnh dài nhất" được. Ở đây, chúng tôi sử dụng "cạnh chiều dài là cạnh có chứa lỗ ở giữa".

Bài toán trở thành: Tìm cạnh có chứa lỗ trong hình góc chính diện top-down.

Bước 1: Tiền xử lý hình ảnh

Gồm các giai đoạn: chuyển đổi sang ảnh gray *cv2.cvtColor()*, làm mờ *cv2.GaussianBlur()*, cân bằng histogram *cv2.equalizeHist()*.

Bước 2: Perspective transform

Tương tự 3.2.1, ta sử dụng perspective transform để biến đổi hình kết quả của bước 1 về góc nhìn chính diện top-down để dễ dàng xác định lỗ.

Bước 3: Threshold

Ta sử dụng threshold để tập trung vào những vùng tối (những vùng có khả năng có lỗ).

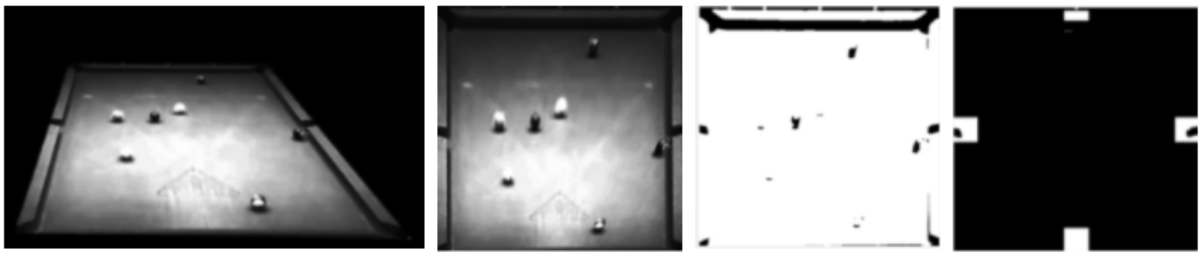
Bước 4: Tạo mask và áp dụng quy tắc xác định lỗ

Ta sử dụng mask để che đi những vùng không cần thiết và chỉ giữ lại 4 vị trí có khả năng có chứa lỗ. Sau đó sử dụng hàm *cv2.bitwise_and()* để kết hợp mask với hình ảnh kết quả của bước 3.

Chúng tôi đặt ra bộ các quy tắc để có thể xác định ô nào có chứa lỗ như sau:

- Nếu có 1 ô hoàn toàn trắng thì ô đó không thể chứa lỗ.
- Lượng màu trắng ở hai ô đối diện nhau cùng nhỏ hơn một ngưỡng và tổng của chúng nhỏ hơn tổng của hai ô còn lại thì hai ô đó có chứa lỗ.
- Ô có ít màu đen hơn trong hai ô đối diện nhau, có lượng màu đen nhiều hơn ô có nhiều màu đen hơn trong hai ô còn lại thì nó có chứa lỗ.
- Lượng màu đen ở hai ô đối diện xấp xỉ nhau và cùng lớn hơn một ngưỡng thì hai ô đó có chứa lỗ.

Ví dụ ở hình 11c), chúng tôi áp dụng quy tắc thứ 4, 2 ô trái/phải có chứa lỗ, nên hai cạnh trái/phải chính là cạnh chiều dài của bàn bida.

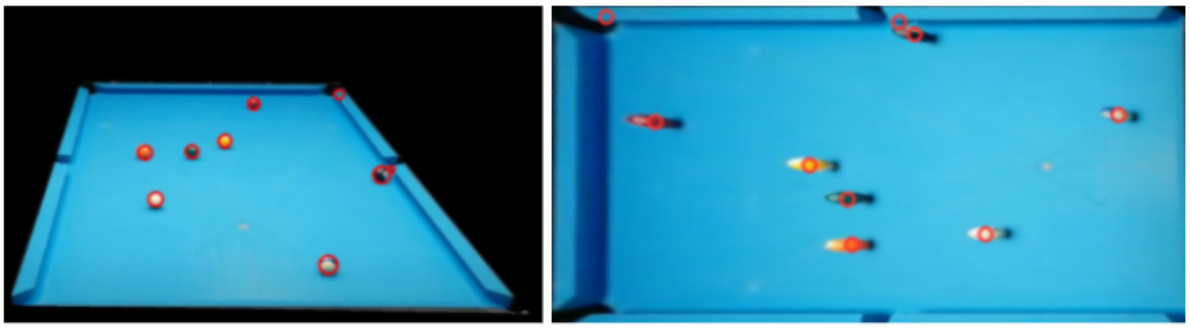


Hình 11: Các giai đoạn để xác định cạnh chiều dài/chiều rộng của bàn bi-da
a) Bước 1: Tiền xử lý. b) Bước 2: Perspective transform.
c) Bước 3: Threshold. d) Bước 4: Áp dụng quy tắc lên mask.

3.2.3. Biến đổi tọa độ của banh sang không gian 3D

Để biến đổi tọa độ của banh từ hình ảnh chụp sang tọa độ không gian 3D, ta sử dụng ma trận biến đổi đã tính được từ bước 3.2.1.

Hàm *cv2.perspectiveTransform()* nhận vào tọa độ 2D của các quả banh đã được phát hiện cùng với ma trận biến đổi, sau đó trả về tọa độ của các quả banh sau khi được biến đổi.



a) b)

Hình 12: Biến đổi tọa độ của banh từ hình chụp sang góc chính diện từ trên xuống.
a) Tọa độ cũ trong hình chụp. b) Tọa độ mới của góc chính diện từ trên xuống.

3.2.4. Loại bỏ những quả banh bị phát hiện sai

Hàm Hough circle transform đôi khi sẽ phát hiện sai ở những vị trí gần lỗ vì lỗ cũng hình tròn, nên để tăng độ chính xác ta cần loại bỏ những quả banh bị phát hiện sai, chính là những quả banh nằm gần lỗ hoặc nằm sát trên cạnh bàn.

Ta ước lượng đường kính của lỗ là 30px, sau đó duyệt qua các quả banh xem quả nào có tọa độ $x < 30$ hoặc $x >$ (width - 30) (tương tự với y) thì ta xóa quả đó ra khỏi danh sách, ta có được kết quả vị trí của banh gần như chính xác tuyệt đối.

a) b)

Hình 13: Loại bỏ banh bị phát hiện sai ở gần lỗ, nằm sát trên cạnh.
a) Trước khi loại bỏ. b) Sau khi loại bỏ.

Cuối cùng, ta đã có được danh sách tọa độ *(x, y)* của các quả banh trong hình, có thể sẵn sàng đưa vào bước mô phỏng game.

4. Mô phỏng

4.1. Định nghĩa bài toán

Với việc xác định được tọa độ của các trái bida, chúng tôi sẽ mô phỏng lên trên một tựa game mà ở đó, ta có thể tương tác với những viên bi, giả lập một trận đấu bida nhằm tìm ra những cách chơi, nước đi mà các bạn mong muốn.

Bằng các thư viện OpenGL, GLM,... chúng ta sẽ cùng nhau tìm hiểu cách sử dụng của các thư viện, các lý thuyết liên quan để render lên các vật thể, mô phỏng các tương tác vật lý, ... Đây cũng là những công đoạn chính để thực hiện nên tựa game này.

4.2. Render

4.2.1. OpenGL

Trước khi bắt đầu, chúng ta sẽ tìm hiểu về OpenGL là gì trước. OpenGL được coi như là một API đa nền tảng, đa ngôn ngữ cho render đồ họa 3D. API được sử dụng để tương tác với GPU nhằm đạt được tốc độ render phần cứng.

Silicon Graphics, Inc. (SGI) Bắt đầu phát triển OpenGL năm 1991 và phát hành nó vào 30 tháng 6 năm 1992. Nó dùng trong các ứng dụng CAD, thực tế ảo, mô phỏng khoa học, mô phỏng thông tin, video game. Từ năm 2006, OpenGL được quản lý bởi consortium công nghệ phi lợi nhuận Khronos Group. Đối thủ chính của OpenGL là DirectX của Microsoft.

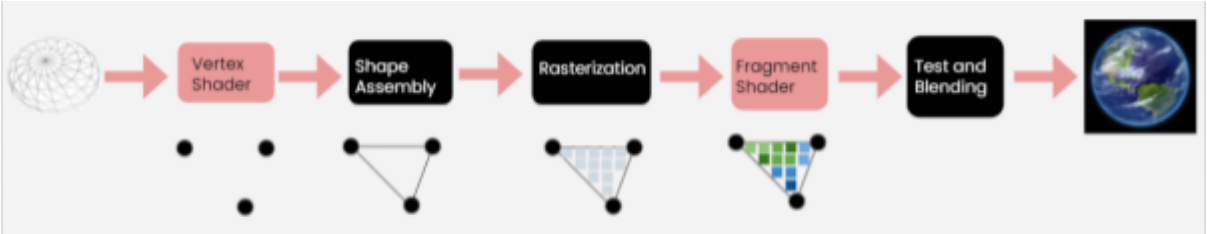
Bởi vì mọi thứ trong OpenGL đều trong không gian 3D, nhưng màn hình hiển thị là 2D gồm một mảng 2 chiều các pixel. Vì vậy một phần quan trọng trong OpenGl là chuyển từ tọa độ 3D sang tọa độ 2D nhằm thể hiện trên màn hình. Quá trình chuyển ấy còn được gọi là graphic pipeline. Quá trình ấy có thể chia thành 2 phần quan trọng như sau:

- Chuyển từ tọa độ 3D sang 2D
- Tô màu cho các pixel hiển thị trên màn hình sao cho phù hợp nhất

Quá trình ấy được thực hiện trên những chương trình nhỏ chạy trên GPU, còn được gọi là shader

4.2.2. Shader

Shader là một chương trình nhỏ, làm việc trực tiếp trên GPU để thực hiện các phép tính phức tạp. Shader được viết bởi ngôn ngữ tương tự C, đó là OpenGL Shader Language (GLSL)



Hình 14: Graphic pipeline

Graphic pipeline gồm nhiều bước. Mỗi bước có những công dụng khác nhau. Đầu ra của bước trước sẽ là đầu vào của bước tiếp theo.

Ở bước đầu của chu trình, Vertex Shader nhận vào tọa độ của các đỉnh. Mục đích chính của bước này là tính toán vị trí các tọa độ đó từ tọa độ 3D này sang tọa độ 3D khác.

Sang bước Shape Assembly, nó nhận tọa độ sau khi tính toán của bước Vertex Shader và mô phỏng lên một hình dạng dựa vào thuộc tính mình cho trước (ví dụ như tam giác, đoạn thẳng hoặc điểm).

Sau bước này, dựa vào hình dạng của đầu ra bước trước, Rasterization sẽ map với các pixel tương ứng trên màn hình, kết quả bước này là tiền đề để thực hiện bước tiếp theo.

Sau khi đã xác định được những pixel cần hiển thị, Fragment Shader sẽ tính toán màu sắc cho từng pixel để hiển thị lên màn hình.

Sau khi qua bước kiểm tra, kết quả ta thu được là một vật thể hoàn chỉnh đã được render lên màn hình.

Tuy quá trình trên phức tạp như vậy, nhưng ta chỉ sẽ qua tâm tới 2 phần cũng là 2 phần quan trọng nhất: Vertex Shader và Fragment Shader.

a. Vertex Shader

Vertex Shader là một trong những shader mà lập trình viên ta có thể lập trình được. Để sử dụng Vertex Shader (cũng tương tự như Fragment Shader), điều đầu tiên ta cần làm là lập trình Shader bằng ngôn ngữ GLSL, đính với chương trình chính của chúng ta, biên dịch nó để có thể sử dụng được.

Vertex Shader nhận vào một mảng các đỉnh và xử lý từng đỉnh riêng biệt (nếu như mảng ấy có 8 đỉnh thì xử lý 8 lần). Những gì Vertex Shader làm là tính toán ra vị trí cuối cùng của từng đỉnh, biến đổi từ vị trí trong tọa độ 3D này sang vị trí trong tọa độ 3D khác (như việc mình di chuyển camera, phóng to, thu nhỏ…). Vertex Shader cũng có trách nhiệm chuẩn bị và xuất ra vài giá trị, biến

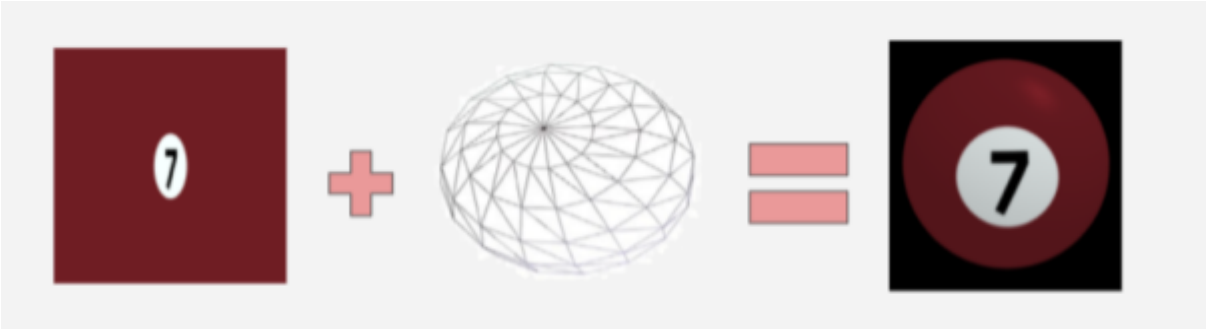
cho Fragment Shader bởi vì chúng ta định nghĩa các biến ở Vertex Shader nhưng không trực tiếp được cho Fragment Shader. Vì vậy, Vertex Shader phải gửi các giá trị, biến tới Fragment Shader.

b. Fragment Shader

Fragment Shader xử lý từng pixel hiện lên trên màn hình. Bên trong Shader này, chúng ta làm việc với mọi thứ liên quan tới bề mặt của vật thể, hiệu ứng phản chiếu, ánh sáng, đổ bóng,... Sản phẩm cuối cùng là màu các pixel.

4.2.3. Object

Các object trong OpenGL được định nghĩa dựa vào tọa độ các đỉnh cấu thành và màu sắc tại mỗi đỉnh. Đối với các vật thể đơn giản, chúng ta có thể dễ dàng khai báo tọa độ đỉnh, màu tại mỗi đỉnh cho chúng như các hình tam giác, chữ nhật hay thậm chí hình lập phương. Nhưng đối với các hình phức tạp và nhiều chi tiết, khai báo vị trí từng đỉnh và màu sắc mỗi đỉnh là tốn thời gian. Vì vậy, nhóm chúng tôi đã sử dụng các phần mềm hỗ trợ - đặc biệt là Blender, nhằm tạo nên những vật thể trong tựa game này



Hình 15: tạo một object

Như vậy, nhóm chúng tôi đã thiết kế toàn bộ các vật thể trong tựa game này. Nhưng để mọi thứ trông chân thật hơn, chúng tôi đã thêm phần ánh sáng vào tựa game này.

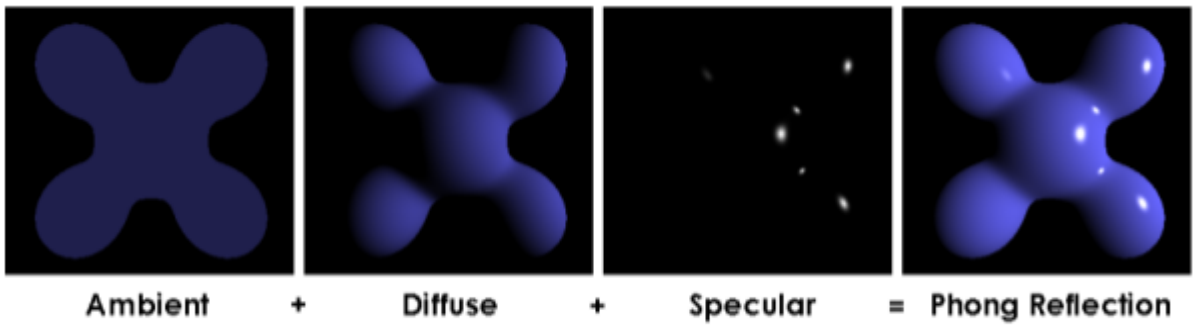
4.2.4. Lighting

Ánh sáng ở ngoài đời cực kỳ phức tạp và phụ thuộc vào nhiều yếu tố, với tài nguyên tính toán của máy tính thì việc mô phỏng như hiện thực là khó đạt được. May mắn, OpenGL có một số model có thể mô phỏng gần giống thực tế nhưng lại đơn giản và dễ hiểu. Một trong những model đó là mô hình chiếu sáng

Phong. Đây cũng là model mà nhóm chúng tôi sử dụng để mô phỏng ánh sáng cho game.

Mô hình chiếu sáng Phong là sự kết hợp giữa 3 thành phần:

- Ambient lighting
- Diffuse lighting
- Specular lighting



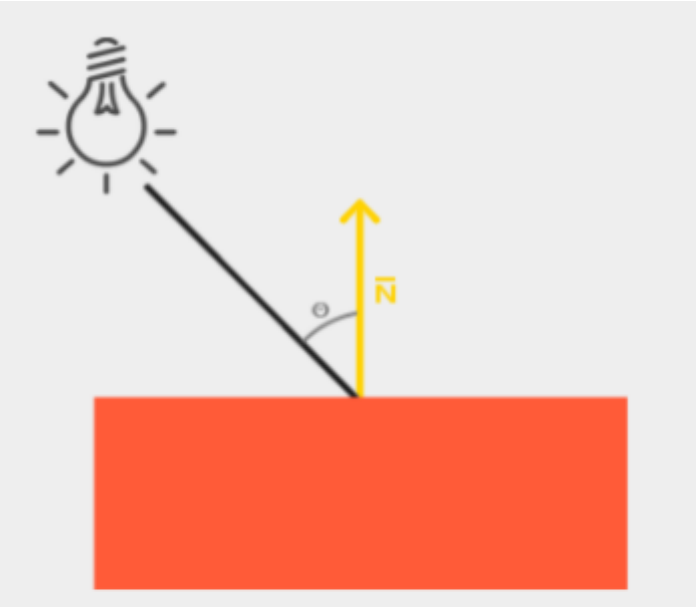
Hình 16: Phong lighting model
Nguồn: https://vi.wikipedia.org/wiki/Phương_pháp_tô_bóng_Phong

a. Ambient lighting

Ambient lighting là mức sáng trung bình, tồn tại trong một vùng không gian. Một không gian lý tưởng là không gian mà tại đó mọi vật đều được cung cấp một lượng ánh sáng lên bề mặt là như nhau, từ mọi phía ở mọi nơi. Nói đơn giản thì Ambient lighting mô tả mức cường độ ánh sáng khắp các bề mặt.

b. Diffuse lighting

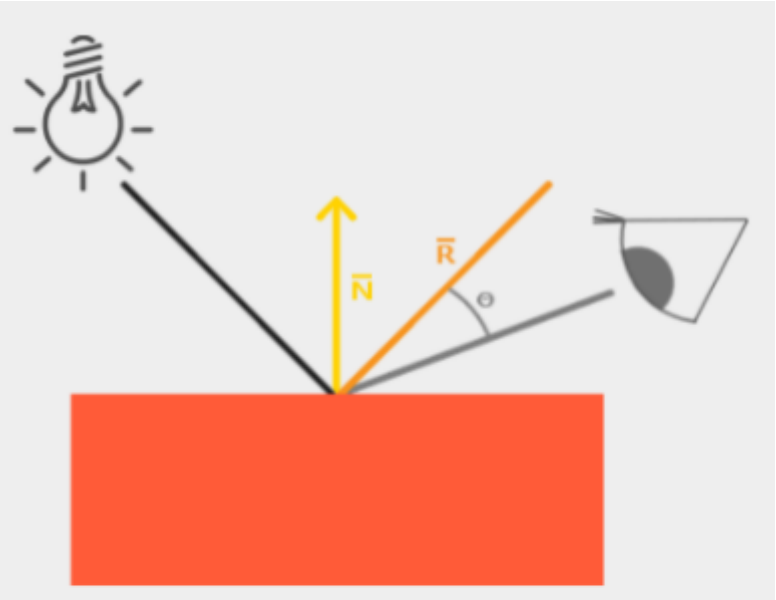
Diffuse lighting cho ta một kết quả thú vị hơn về việc chiếu sáng. Bằng việc xác định vectơ pháp tuyến của các bề mặt vật thể, cùng với vector chỉ hướng chiếu của nguồn sáng lên vật thể, độ sáng của từng vị trí sẽ khác nhau dựa vào góc giữa 2 vectơ trên



Hình 17: Diffuse lighting
Nguồn: https://learnopengl.com/

c. Specular lighting

Tương tự như diffuse lighting, specular lighting cũng dựa vào hướng chiếu sáng và vecto pháp tuyến của bề mặt. Ngoài ra, thành phần này còn có thêm hướng mà người dùng đang nhìn tới. Vị trí mà ta nhìn sẽ là sáng nhất nếu như trùng với ánh sáng phản xạ trên mặt phẳng.



Hình 18: Specular lighting
Nguồn: https://learnopengl.com/

Như vậy, khi kết hợp 3 thành phần trên, kết quả sẽ là các vật phẩm đã được tô sáng, khiến chúng trông chân thật hơn.

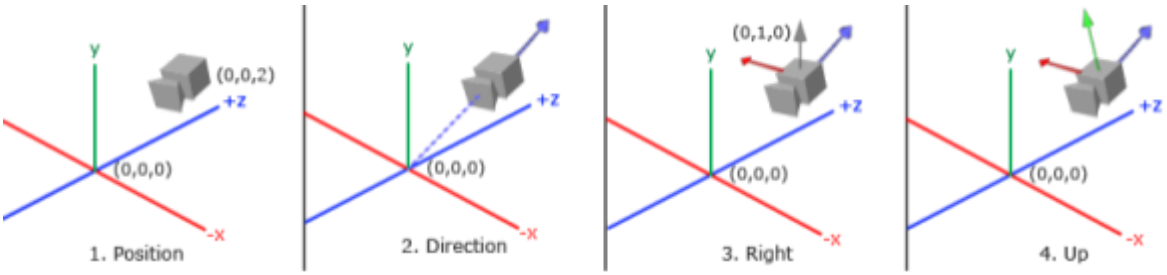
4.2.5.Camera

Qua một quá trình, ta đã có được một không gian chứa các vật thể. Nhưng để quan sát được các vật thể đó, ta cần một thành phần như một chiếc Camera, có thể di chuyển khắp không gian ấy để dễ dàng thấy được xung quanh.

Việc điều khiển camera đồng nghĩa với việc thay đổi góc nhìn so với vị trí ban đầu. Để làm được điều đó, ta sẽ xài các phép biến đổi ma trận để tìm ra vị trí của các vật thể trên màn hình sau khi điều khiển camera (điều khiển góc nhìn).

Ta định nghĩa Camera gồm các thành phần:

- Vị trí của nó trong không gian chứa các vật thể.
- Vector chỉ hướng nhìn của Camera
- Vector hướng lên của Camera
- Vector hướng phải của Camera



Hình 19: Các thành phần của Camera
Nguồn: https://learnopengl.com/

3 vecto này tạo nên một hệ trục tọa độ với tâm là Camera.

Ta có thể thiết lập chuột và bàn phím để thay đổi tọa độ tâm, điều chỉnh các vector nhằm thay đổi góc nhìn giống như việc chúng ta đang di chuyển camera.

Để xác định được tọa độ các vật thể trong không gian ứng với góc nhìn Camera. ta định nghĩa ma trận sau:

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hình 20: Ma trận thay đổi góc nhìn
Nguồn: https://learnopengl.com/

Trong đó:

- R là vector hướng phải của Camera.
- U là vector hướng lên của Camera.
- D là vector hướng nhìn của Camera.
- P là tọa độ của Camera.

Như thế, bằng phép biến đổi dưới đây, ta sẽ có được tọa độ của các vật thể trong không gian 3D mới ứng với góc nhìn của camera:

$$\begin{pmatrix} x_{\text{view}} \\ y_{\text{view}} \\ z_{\text{view}} \\ 1 \end{pmatrix} = LookAt * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Hình 21: công thức tính tọa độ sau khi thay đổi góc nhìn
Nguồn: https://learnopengl.com/

4.3. Mô phỏng vật lý

4.3.1. Các đặc tính của một viên bi

Đối với mỗi trái banh, nhóm chúng tôi quy định các đặc tính sau:

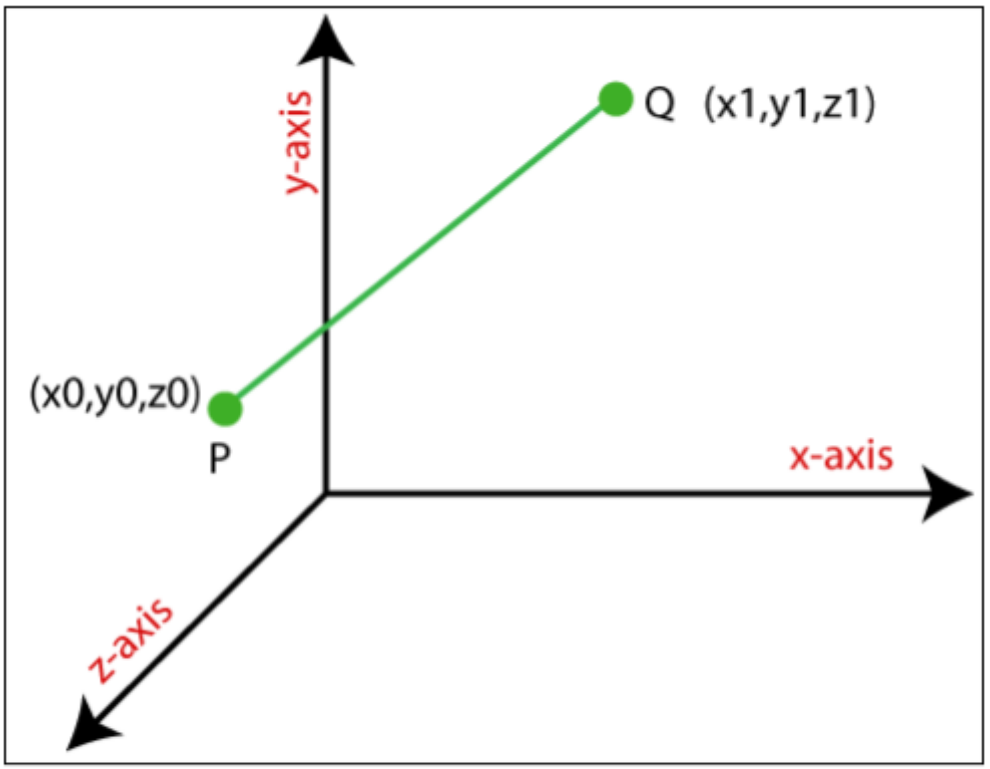
- Là một vật thể và được tạo nên các đỉnh như đã đề cập ở 4.2.3.
- Có giá trị bán kính là cố định và giống nhau.
- Có khối lượng là cố định và giống nhau.
- Có các hệ số ma sát lăn, ma sát trượt cụ thể.
- Có vectơ chỉ hướng di chuyển (được chuẩn hóa).
- Có giá trị độ lớn của vận tốc.
- Vectơ vận tốc = giá trị độ lớn × vectơ chỉ hướng

- Có giá trị độ lớn của gia tốc
- Có tác động bởi trọng lực g=9.8m/s^2

4.3.2. Chuyển động của bi

Đối với viên bi, sau khi bị tác động lực, viên bi sẽ có 2 chuyển động

a. Tịnh tiến



Hình 22: tịnh tiến một điểm
Nguồn: https://www.tutorialandexample.com/

Tịnh tiến viên bi tại vị trí này sang vị trí khác đồng nghĩa với tịnh tiến đồng thời các điểm tạo nên viên bi ấy. Bằng phép biến đổi ma trận như sau, ta có thể tịnh tiến viên bi theo một vectơ T cho trước:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$

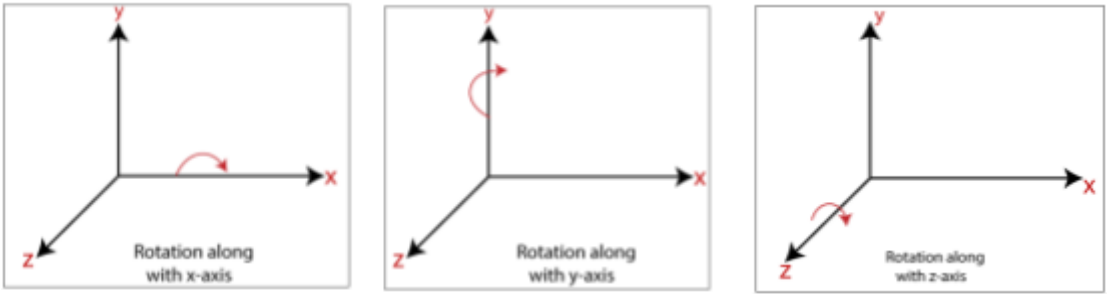
Hình 23: công thức tìm tọa độ sau khi tịnh tiến
Nguồn: https://learnopengl.com/

b. Xoay

Đối với chuyển động xoay thì phức tạp hơn. Đối với chuyển động tịnh tiến ta chỉ cần xác định vecto tịnh tiến, thì với chuyển động này, ta cần xác định 2 thành phần quan trọng như sau:

- Trục xoay/ vecto
- Góc xoay

Khi ta nói viên bi xoay, đồng nghĩa với tất cả các điểm tạo thành viên bi xoay xung quanh 1 trục theo 1 góc θ cho trước



Hình 24: xoay một điểm quanh các trục tọa độ
Nguồn: https://www.tutorialandexample.com/

Tương ứng, ta có tọa độ các điểm sau khi xoay bằng phép biến đổi:

- Xoay quanh trục x một góc θ:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

Hình 25: xoay một điểm quanh trục x
Nguồn: https://learnopengl.com/

- Xoay quanh trục y một góc θ:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

Hình x: xoay một điểm quanh trục y
Nguồn: https://learnopengl.com/

- Xoay quanh trục z một góc θ:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

Hình 26: xoay một điểm quanh trục z
Nguồn: https://learnopengl.com/

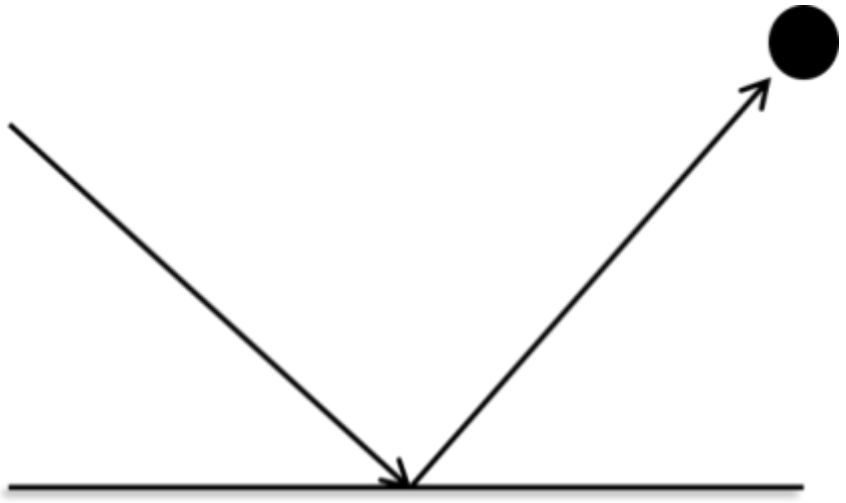
Tổng quát, tọa độ của điểm sau khi xoay quanh trục xoay/ vectơ (Rx,Ry,Rz) (vecto đã chuẩn hóa) một góc θ được tính bằng công thức:

$$\begin{bmatrix} \cos\theta + R_x{}^2(1-\cos\theta) & R_xR_y(1-\cos\theta) - R_z\sin\theta & R_xR_z(1-\cos\theta) + R_y\sin\theta & 0 \\ R_yR_x(1-\cos\theta) + R_z\sin\theta & \cos\theta + R_y{}^2(1-\cos\theta) & R_yR_z(1-\cos\theta) - R_x\sin\theta & 0 \\ R_zR_x(1-\cos\theta) - R_y\sin\theta & R_zR_y(1-\cos\theta) + R_x\sin\theta & \cos\theta + R_z{}^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Hình 27: xoay một điểm quanh vectơ bất kì
Nguồn: https://learnopengl.com/

Lưu ý: (x, y, z) là tọa độ của điểm trước khi biến đổi

4.3.3. Bi chạm bàn



Hình 28: bi va chạm thành bàn

Ta định nghĩa bi va chạm thành bàn khi khoảng cách từ tâm bi tới thành bàn bé hơn bán kính viên bi.

Khi viên bi va chạm thành bàn, viên bi sẽ di chuyển theo một hướng/vectơ khác. Để tìm được vectơ này, ta đi qua 2 bước sau:

- Tìm vectơ đối xứng với vectơ mà viên bi di chuyển trước khi chạm thành bàn qua mặt phẳng vuông góc với thành bàn ấy.
- Lấy vectơ ngược chiều với vectơ trên.

4.3.4. Thụt bi trắng

Thụt bi trắng hay tác động lực lên viên bi đang đứng yên (lực tác động là xuyên tâm bi) sẽ tạo ra hiện tượng:

- Bi sẽ trượt (không lăn) trong một khoảng thời gian ngắn.
- Sau khoảng thời gian trên, bi sẽ lăn cho tới khi dừng hẳn.

Cụ thể:

- Sau khi tác động một lực F cho trước (tác động tức thời), viên bi chuyển động tịnh tiến với độ lớn vận tốc là v_{max} .
- Do có momen, viên bi bắt đầu xoay, nhưng xoay chậm.
- Do ban đầu, tốc độ xoay không đồng bộ với tốc độ v_{max} nên viên bi sẽ trượt.
- Tốc độ xoay thì càng tăng còn tốc độ tịnh tiến càng giảm (do tác động của ma sát trượt nên viên bi chuyển động chậm dần đều), đến khi cả 2 tốc độ bằng nhau thì dừng quá trình trượt và chuyển sang trạng thái lăn.

4.3.5. Bi chạm bi

Ta định nghĩa viên bi thứ 1 chạm viên bi thứ 2 nếu như khoảng cách giữa chúng bé hơn tổng 2 bán kính viên bi.

Khi 2 viên bi va chạm nhau sẽ làm thay đổi hướng di chuyển của từng viên, cụ thể:

$$\mathbf{v}_1' = \mathbf{v}_1 - \frac{2m_2}{m_1+m_2}\,\frac{\langle \mathbf{v}_1-\mathbf{v}_2,\,\mathbf{x}_1-\mathbf{x}_2\rangle}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,(\mathbf{x}_1-\mathbf{x}_2),$$
$$\mathbf{v}_2' = \mathbf{v}_2 - \frac{2m_1}{m_1+m_2}\,\frac{\langle \mathbf{v}_2-\mathbf{v}_1,\,\mathbf{x}_2-\mathbf{x}_1\rangle}{\|\mathbf{x}_2-\mathbf{x}_1\|^2}\,(\mathbf{x}_2-\mathbf{x}_1)$$

Hình 29: vectơ vận tốc sau va chạm của 2 viên bi
Nguồn: https://en.wikipedia.org/wiki/Elastic_collision

Trong đó:

- v_1, v_2 lần lượt là vectơ vận tốc trước va chạm của bi thứ 1, bi thứ 2

- v_1', v_2' lần lượt là vectơ vận tốc sau va chạm của bi thứ 1, bi thứ 2
- x_1, x_2 lần lượt là tọa độ tâm của bi thứ 1, bi thứ 2
- m_1, m_2 là khối lượng bi

Chương 3:
KẾT QUẢ VÀ THẢO LUẬN

Ở chương này, nhóm chúng tôi sẽ tiến hành thực hiện đánh giá các phương pháp đã tìm hiểu ở chương trước, từ đó so sánh và đưa ra kết luận cho phương pháp cuối cùng được sử dụng trong sản phẩm.

1. Chuẩn bị đánh giá

1.1. Dữ liệu đánh giá

Nhóm chúng tôi chuẩn bị 127 hình ảnh chụp các trận đấu bida với nhiều góc nhìn khác nhau (trong đó có 58 tấm chụp từ các quán bida, các tấm còn lại thu thập từ các video về các trận đấu chuyên nghiệp và trận đấu trong game trên Youtube).

Đối với bài toán định vị góc bàn, nhóm chúng tôi xác định 4 cạnh bàn bida bằng công cụ gán nhãn ImageTagger [14]. Việc này giúp chúng tôi đánh giá xem phương pháp tiếp cận và thuật toán định vị góc bàn của chúng tôi hiệu quả hay không.

Còn với bài toán phát hiện banh, nhóm chúng tôi cũng xác định vị trí, kích thước từng trái banh bằng công cụ gán nhãn LabelImg [15]. Đây cũng là cơ sở để nhóm chúng tôi đánh giá các hướng tiếp cận, thuật toán và chọn ra phương pháp tối ưu.

1.2. Thang đo đánh giá

1.2.1. Intersection over Union

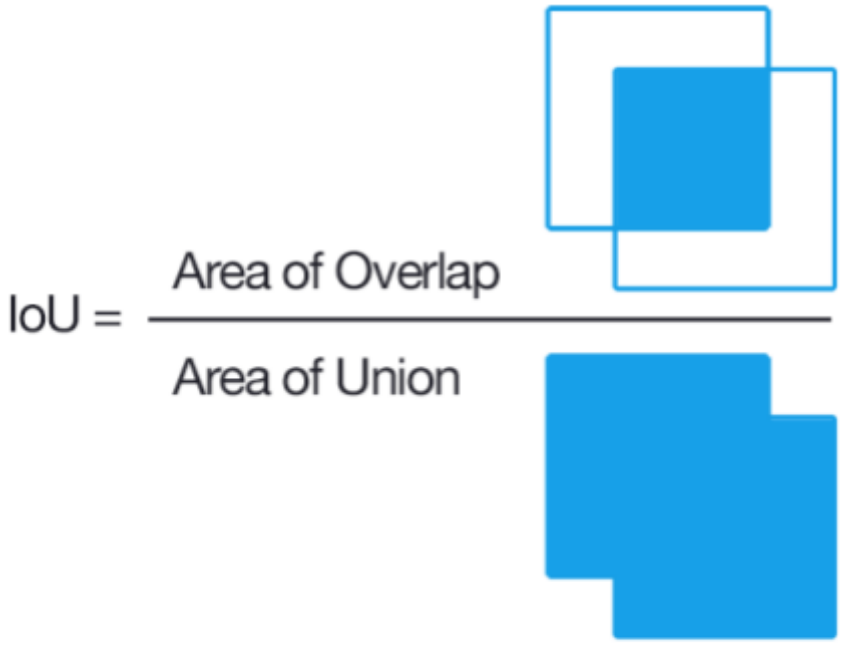
Intersection over Union (IoU) là chỉ số đánh giá được sử dụng để đo độ chính xác của phát hiện đối tượng trên tập dữ liệu cụ thể.

Để áp dụng IoU vào đánh giá, ta cần:

- Nhãn dữ liệu: là đường bao (bounding box) mà chúng tôi đã gán nhãn trước.

- Kết quả dự đoán: là đường bao mà sau khi chúng tôi áp dụng các thuật toán để xác định.

Chỉ số IoU là tỉ lệ giữa đo lường mức độ giao nhau giữa hai đường bao để nhằm xác định hai khung hình có bị đè chồng lên nhau không. Tỷ lệ này được tính dựa trên phần diện tích giao nhau giữa 2 đường bao với phần tổng diện tích giao nhau và không giao nhau giữa chúng.

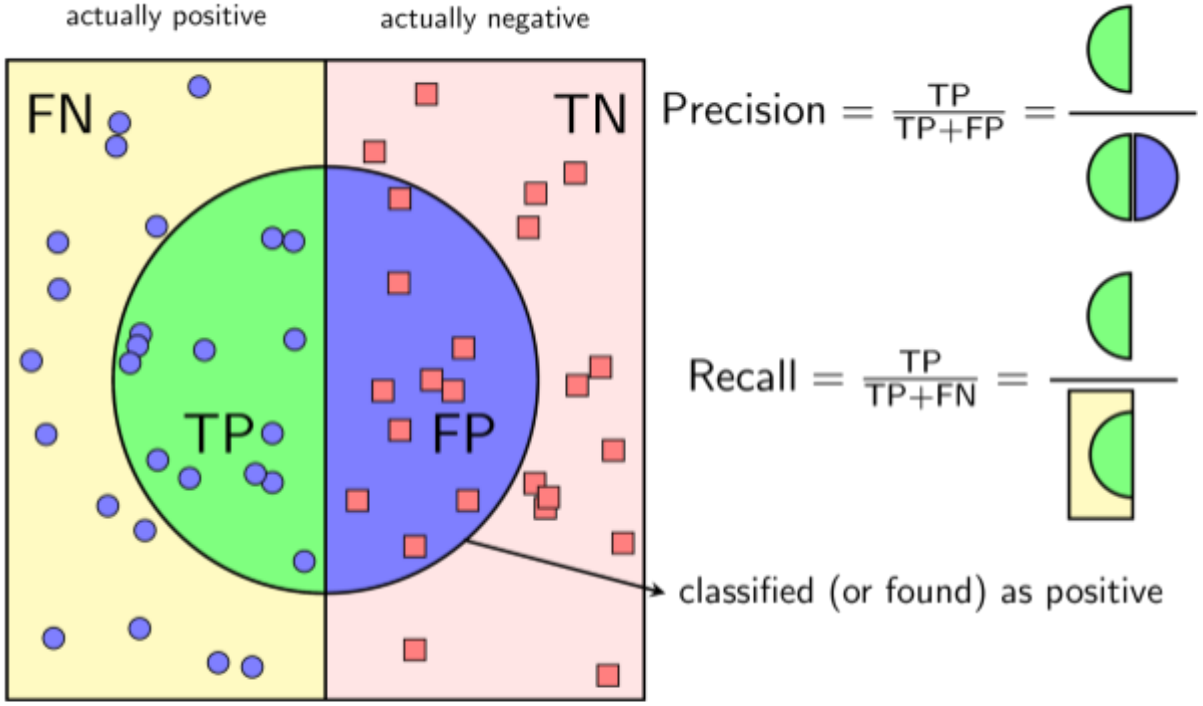


Hình 30: chỉ số IoU
Nguồn: http://techshare247.com/

1.2.2. Mean Average Precision

Trước khi nói về Mean Average Precision (mAP), nhóm chúng tôi giới thiệu về Precision và Recall

- Precision: Đánh giá độ tin cậy của kết luận đưa ra
- Recall: Đánh giá khả năng tìm kiếm toàn bộ các ground truth của mô hình



Hình 31: Precision và Recall trong phân lớp nhị phân
Nguồn: https://machinelearningcoban.com/

- Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao.
- Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỷ lệ bỏ sót các điểm thực sự positive là thấp.
- AP (Average Precision) là phần diện tích phía dưới đường biểu diễn precision-recall bằng cách tính tổng các hình chữ nhật xấp xỉ.

Như vậy, mAP là trung bình của AP từng ngưỡng (đôi khi cũng chính là AP).

2. Đánh giá

2.1. Định vị góc bàn

Đối với bài toán định vị góc bàn, chúng tôi sử dụng thang đo IoU để tính chênh lệnh diện tích giữa đa giác tìm được và đa giác được gán nhãn trong bộ dữ liệu. Ngoài ra, để đánh giá được mức độ hiệu quả của thuật toán khi tích hợp vào một ứng dụng hoặc trò chơi, cũng cần so sánh tốc độ thực thi, hay cụ thể ở đây chúng tôi sử dụng FPS để đánh giá số hình mà mỗi phương pháp có thể xử lí trên một giây.

Dưới đây là bảng đánh giá cho bốn phương pháp được đề cập ở chương trước:

PHƯƠNG PHÁP	IoU	FPS
Contour + ngưỡng cố định	0.800	**18.79**
Contour + ngưỡng linh hoạt	0.831	14.29
Bao lồi + ngưỡng cố định	0.805	16.24
Bao lồi + ngưỡng linh hoạt (Phương pháp được chọn)	**0.903**	12.66

Bảng 1: Kết quả đánh giá cho bài toán định vị góc bàn.

Thông qua số liệu đánh giá trong Bảng 1, có thể thấy rằng phân vùng dựa trên ngưỡng linh hoạt sẽ cho ra kết quả tốt hơn so với ngưỡng cố định. Ngoài ra, việc sử dụng thuật toán bao lồi cho bài toán ước lượng đa giác cũng đạt được kết quả tốt hơn so với phương pháp phát hiện contour. Ở đây, nhóm chúng tôi sử dụng phương pháp phân vùng bằng ngưỡng linh hoạt, kết hợp với ước lượng đa giác bằng thuật toán bao lồi cho sản phẩm cuối cùng, vì phương pháp này mang lại độ chính xác cao nhất, đồng thời tốc độ thực thi cũng nằm ở mức chấp nhận được với 12.66 ảnh được xử lý trên giây.

2.2. Phát hiện banh bi-da

Ở bài toán phát hiện banh bi-da với nhiều vật thể, chúng tôi sử dụng thang đo mAP với ngưỡng IoU = 0.5 nhằm đánh giá được mức độ chênh lệch giữa kết quả của phương pháp so với dữ liệu được gán nhãn.

Phương pháp	Thang đo mAP@0.5
Canny edge detection + Contour	0.21
Hough circle transform	0.75
Canny edge detection + Hough circle transform	0.65

Bảng 2: Kết quả đánh giá cho bài toán phát hiện banh bi-da.

Canny edge detection + Contour

Phương pháp này cho ta độ chính xác thấp. Nguyên nhân là do viền giữa hình tròn có chứa số và phần màu của quả banh, cũng như viền giữa phần màu trắng

và phần màu của banh sọc quá rõ ràng, nên được phát hiện vào loại "strong edge". Do đó 1 quả banh có thể được phát hiện lên đến 3 lần. Phương pháp này không phù hợp để áp dụng cho bài toán phát hiện banh bi-da.

Hough circle transform

Đây là phương pháp cho ta độ chính xác cao nhất. Vì hàm Hough làm việc rất tốt trên các bài toán phát hiện vật thể trình tròn. Ta chỉ cần điều chỉnh bộ tham số cho phù hợp là đã có thể tăng độ chính xác lên rất nhiều.

Canny edge detection + Hough circle transform

Cũng tương tự như dùng Contour, thì Canny edge detection phát hiện sai những đường viền trong quả banh thành cạnh, nên hàm Hough cũng nhận diện sai những hình tròn trong hình, làm giảm độ chính xác.

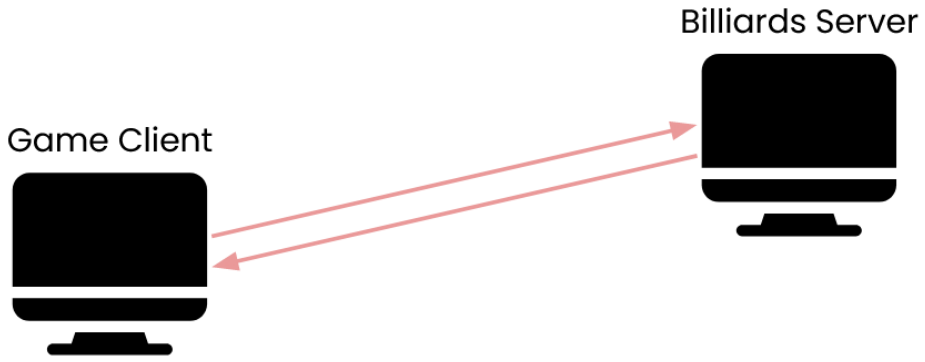
Kết luận

Nhóm chúng tôi đã sử dụng phương pháp **Hough Circle Transform** (có độ chính xác cao nhất) áp dụng vào đồ án lần này. Và khi kết hợp với bước loại bỏ những banh bị phát hiện sai (ở phần 3.2.4), đã tăng độ chính xác từ 0,75 lên thành **0,78**. Tuy chênh lệch không nhiều nhưng điều này mang lại ý nghĩa rất lớn, vì nếu giữ nguyên banh ở các vị trí ngay lỗ hay nằm trên cạnh thì sẽ bị xung đột với các bước mô phỏng vật lý ở trong game.

Chương 4:
XÂY DỰNG ỨNG DỤNG MINH HỌA

1. Tổng quan

Sau khi đã tìm hiểu và đánh giá một số phương pháp được đề cập ở các phần trước, nhóm chúng tôi đã cài đặt được hai mô-đun hoạt động độc lập nhau: mô-đun xử lý hình ảnh được hiện thực thông qua thư viện OpenCV bằng ngôn ngữ Python, và mô-đun mô phỏng trò chơi Bi-da được hiện thực thông qua thư viện OpenGL bằng ngôn ngữ C++.



Hình 32: Mô hình máy chủ - máy khách.

Để kết hợp giữa hai mô-đun độc lập của hai ngôn ngữ khác nhau, chúng tôi đã thiết kế mô hình máy chủ - máy khách (server - client) cho phép các mô-đun giao tiếp với nhau (Hình 32). Trong đó, mô-đun mô phỏng trò chơi đóng vai trò như máy khách, cho phép người dùng trực tiếp điều khiển và tương tác với trạng thái trò chơi hiện tại; mô-đun xử lý hình ảnh đóng vai trò như một máy chủ, tiếp nhận đầu vào là đường dẫn đến một bức ảnh trên máy tính, đầu ra trả về là tọa độ của các quả banh bi-da cho máy khách mô phỏng.

Để thiết kế máy chủ, chúng tôi sử dụng thư viện Flask của ngôn ngữ Python để tiếp nhận và xử lý các yêu cầu HTTP từ máy khách gửi đến. Ở phía máy khách, một thư viện mã nguồn mở [12] được sử dụng để gửi yêu cầu đến máy chủ.

2. Cài đặt

Sản phẩm của nhóm chúng tôi được lưu trữ trên Github ở địa chỉ [13]. Sản phẩm có thể được tải về và cài đặt theo quy trình hướng dẫn bên dưới.

2.1. Cài đặt trên hệ điều hành Windows

Yêu cầu kỹ thuật:

- Windows 32-bit hoặc 640bit.
- Trình biên dịch GNU dành cho Windows: MinGW, MinGW-w64, hoặc TDM-GCC.
- GCC phiên bản 5.1.0 trở lên.
- OpenGL phiên bản 3.3 trở lên.
- Python 3 + pip.

Tiến hành cài đặt:

Đối với người dùng cuối: Để thực hiện quy trình cài đặt hoàn chỉnh, thực thi lệnh `make install`. Sau khi thực thi thành công, tệp tin máy khách `BilliardsGame.exe` sẽ được tạo. Để khởi động máy chủ, vui lòng đến bước tiếp theo.

Đối với các nhà phát triển: Để giữ lại các tệp tin mã máy sau khi biên dịch, sử dụng lệnh `make`.

Ngoài ra, nhằm tăng tính tiện dụng cho quá trình phát triển và kiểm thử chương trình, có thể thực hiện lệnh `make run` để biên dịch và lập tức khởi động GameClient.

Sau khi hoàn tất việc lập trình và kiểm thử mã nguồn, thực hiện lệnh `make clean` để dọn dẹp những tệp tin không cần thiết.

Khởi động máy chủ

Cài đặt các thư viện Python cần thiết thông qua lệnh pip:

```
pip install -r requirements.txt
```

Khởi động máy chủ bằng lệnh: `python3 server/src/server.py`.

2.2. Cài đặt cho các hệ điều hành Linux

Yêu cầu kỹ thuật:

- Docker Engine
- Docker Compose

Tự xây dựng Docker Image:

Để xây dựng lại Docker Image, di chuyển đến thư mục chứa chương trình và thực hiện lệnh sau:

```
docker-compose build
```

Khởi động trò chơi:

Để khởi động cả máy chủ và máy khách, di chuyển đến thư mục chứa chương trình và thực hiện lệnh

```
docker-compose up
```

3. Hướng dẫn sử dụng

3.1. Điều khiển

Sau khi đã khởi động được máy chủ và máy khách, có thể bắt đầu chơi game theo hướng dẫn dưới đây:

- Khi mới bắt đầu game, người chơi có thể sử dụng các phím WASD và chuột để có thể di chuyển tùy ý trong không gian game.
- Người chơi có thể sử dụng lăn chuột để phóng to, thu nhỏ.
- Sau khi sử dụng Camera, người chơi có thể nhấn nút C để khóa vị trí nhìn tại bi trắng cũng là viên bi mà chúng ta sẽ tác động lực vào.
- Có các mức lực là các số từ 1 tới 5 tương ứng lực tác động từ nhẹ tới mạnh.
- Sau khi xác định được mức lực, xác định hướng bắn và nhấn Space.
- Để chơi lại từ đầu, nhấn phím P.

3.2. Mô phỏng trạng thái Billiards

Để mô phỏng trạng thái game từ một ảnh đầu vào, thực hiện thao tác kéo thả ảnh từ ngoài giao diện Desktop máy tính vào cửa sổ trò chơi. Tọa độ của các quả bóng sẽ được cập nhật thành trạng thái bạn mong muốn.

Chương 5:
ƯU ĐIỂM, HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN

1. Các dự án liên quan

Trong quá trình khảo sát thực hiện đề tài này, nhóm chúng tôi có tìm hiểu và đánh giá một số dự án liên quan nhằm rút ra các điểm mạnh và điểm thiếu sót của từng hướng tiếp cận. Dưới đây là tổng hợp một số dự án liên quan đến mô phỏng trò chơi Bi-da.

- Dự án của Richárd Bodai [8] có giao diện điều khiển trực quan dễ sử dụng, có mô phỏng vật lí tương đối tốt, tuy nhiên phần render vẫn chưa chân thực khi chưa có đổ bóng cho vật thể.
- Dự án của Xufeng Wu [9] có render chân thật với khả năng tô bóng và đổ bóng và hệ thống vật lí tương đối tốt. Tuy nhiên vẫn còn thiếu khung cảnh xung quanh nên không gian còn hơi hướng u ám.
- Dự án của [10] được render hoàn toàn bằng code, không sử dụng mô hình tự tạo bên ngoài. Điều này cho phép game được thực thi với hiệu suất cao hơn. Tuy nhiên game có góc nhìn 2D và không đáp ứng được yêu cầu của bài toán ban đầu.
- Dự án của [11] là một dự án lớn, được thực hiện chỉn chu với báo cáo chi tiết tương đối đầy đủ nhưng vẫn cần bổ sung thêm ở công đoạn render. Trò chơi được mô phỏng với giao diện điều khiển dễ sử dụng, có cảnh vật và khung cảnh căn phòng xung quanh. Tuy nhiên hệ thống vẫn còn thiếu sót đổ bóng cho vật thể và hệ thống vật lí vẫn chưa gần gũi với thực tế.

Theo khảo sát của nhóm chúng tôi, hiện vẫn chưa có các sản phẩm kết hợp được giữa hai phần xử lý hình ảnh và mô phỏng trò chơi để đưa trạng thái của các ván đấu bi-da ngoài đời vào môi trường ảo.

2. Ưu điểm và thiếu sót của đồ án

Trong sản phẩm của chúng tôi, mô hình chiếu sáng Phong được áp dụng nhằm tô bóng chân thực cho vật thể. Về hệ thống vật lí, chúng tôi đã tham khảo các công thức va chạm đàn hồi, kết hợp với lực ma sát trượt và ma sát lăn nhằm mô phỏng chính xác hơn cho các chuyển động vật lí của banh bi-da khi người dùng tương tác. Ngoài ra, các mô hình 3D đơn giản như banh bi-da, bóng đèn hoặc căn phòng cũng được render bằng code nhằm tối ưu hiệu suất của trò chơi.

Về mặt xử lý hình ảnh, chúng tôi đã thực hiện so sánh các phương pháp được đề xuất để phân vùng và phát hiện vật thể nhằm thiết kế được một hệ thống cân bằng giữa mức độ hiệu quả và tốc độ thực thi của chương trình.

Bên cạnh các điểm mạnh trên, đồ án của chúng tôi vẫn còn tồn đọng nhiều thiếu sót. Chẳng hạn như trò chơi bi-da vẫn chưa mô phỏng được đổ bóng cho các vật thể, và trò chơi vẫn chưa được thiết kế các luật để người dùng có thể chơi và tính điểm như một ván đấu bi-da ngoài đời.

Ngoài ra, vì giới hạn về mặt thời gian thực hiện cũng như phạm vi môn học, chúng tôi vẫn chưa đánh giá mức độ hiệu quả của các phương pháp tân tiến hiện này sử dụng các mạng học sâu trên bài toán phân vùng và bài toán phát hiện vật thể. Chúng tôi tin rằng việc thử nghiệm và đánh giá các phương pháp đó sẽ giúp đồ án này được hoàn chỉnh hơn.

3. Hướng phát triển

Trong tương lai, nhóm chúng tôi dự định phát triển đồ án này dựa trên các hướng sau:

- Phát triển thành một trò chơi hoàn chỉnh với hệ thống chấm điểm và ghi lỗi cho người chơi.
- Hỗ trợ việc chơi trực tuyến kết nối giữa nhiều người với nhau.
- Thiết kế một hệ thống Trí tuệ Nhân tạo để học cách chơi bi-da từ môi trường mô phỏng.
- Thử nghiệm và đánh giá các phương pháp học sâu trên bài toán phân vùng và bài toán phát hiện vật thể.
- Phát triển mô-đun xử lý ảnh thành xử lý trên dữ liệu video để máy tính phân tích một trận đấu bi-da ngoài đời và đóng vai trò như trọng tài để tính điểm.

TÀI LIỆU THAM KHẢO

1. Segmentation with Gaussian mixture models. https://scipy-lectures.org/advanced/image_processing/auto_examples/plot_GMM.html
2. Satoshi Suzuki, KeiichiA be (1985). Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, Volume 30, Issue 1, 1985, Pages 32-46, ISSN 0734-189X.
3. Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, 10*(2), 112–122. doi:10.3138/fm57–6770-u75u-7727
4. Socret Lee. Simplify Polylines with the Douglas Peucker Algorithm.
5. Hough Transform with OpenCV. https://learnopencv.com/hough-transform-with-opencv-c-python/
6. Perspective transformation in Python OpenCV. https://www.geeksforgeeks.org/perspective-transformation-python-opencv/
7. OpenGL Tutorial. http://www.opengl-tutorial.org/
8. Richárd Bodai. Billiard Game - OpenGL. https://youtu.be/6uzf1nXZW6E
9. Xufeng Wu. Simple 3D animation with OpenGL. https://youtu.be/xRLHye-X5-s
10. AB Creative Studio. Pool Table in openGL. https://youtu.be/JFPuf2zC-Po

11. Quoc-Tuan Truong, Thac-Thong Nguyen, giaplv57. 3D Billiards Game OpenGL.
https://github.com/tqtg/3D-Billiards-Game-OpenGL

12. Elviss Strazdins, Nick-I. A. Single-header C++ HTTP request class.
https://github.com/elnormous/HTTPRequest

13. UIT Billiards Club. Mô phỏng Trạng thái Game Billiards.
https://github.com/khanh-moriaty/billiards-game

14. Fiedler, Niklas and Bestmann, Marc and Hendrich, Norman (2018). ImageTagger: An Open Source Online Platform for Collaborative Image Labeling. RoboCup 2018: Robot World Cup XXII. Springer.

15. Tzutalin (2015). LabelImg.
https://github.com/tzutalin/labelImg