TRƯỜNG ĐH KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ TRI THỨC

oOo

LUẬN VĂN TỐT NGHIỆP CỬ NHÂN TIN HỌC KHÓA 99

ĐỀ TÀI

NGHIÊN CỨU CÁC THUẬT TOÁN TẠO BÓNG TRONG ĐỒ HỌA BA CHIỀU TƯƠNG TÁC THỜI GIAN THỰC

HƯỚNG DẪN : Th.S ĐINH NGUYỄN ANH DŨNG
THỰC HIỆN : NGUYỄN VĂN THÀNH 9912072
NGUYỄN THANH SƠN 9912062

Tp. Hồ Chí Minh, 7/2003

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn tới các thầy cô khoa Công nghệ thông tin trường Đại học Khoa học tự nhiên, những người đã ân cần dạy dỗ cho chúng em những kiến thức bổ ích và quý giá trong suốt 4 năm học qua, những người đã trang bị cho chúng em hành trang quý giá để bước vào đời

Chúng em xin gửi lời cảm ơn sâu sắc tới thầy Đinh Nguyễn Anh Dũng, người đã tận tình chỉ bảo và hướng dẫn chúng em thực hiện tốt luận văn tốt nghiệp này

Chúng em xin gửi lời cảm ơn tới gia đình và bạn bè, hậu phương vững chắc cho tiền tuyến chúng em trong suốt những năm học gian khổ, và gần đây đã cho chúng em nguồn động viên to lớn về tinh thần và vật chất để chúng em có thể hoàn thành tốt luận văn tốt nghiệp này

Chúng em xin gửi lời cảm ơn tới sự giúp đỡ của anh Trần Thế Vinh dành cho chúng em trong thời gian thực hiện luận văn này

MỤC LỤC

PHẦN 1 : MỞ ĐẦU

Trong thực tế, con người cảm nhận thế giới bằng các giác quan của mình. Một vật thể có thể được cảm nhận bằng các xúc giác qua sự sờ mó hay được cảm nhận bằng mùi qua khứu giác , tuy nhiên trong một chừng mực nào đó có thể nói cảm nhận vật thể đó bằng thị giác qua màu sắc, đặc điểm, hình dạng, … sẽ cho con người một cảm nhận đầy đủ, trực quan và rõ ràng nhất. Vì vậy nếu có thể xây dựng được các chương trình trên máy tính mô phỏng được các vật thể, hiện tượng trong thế giới thực thì sẽ cung cấp cho người dùng một cách tiếp cận bằng thị giác trực quan hơn về các vấn đề mà họ đang xem xét.

Với công nghệ phần cứng máy tính hiện nay, các hạn chế cơ bản về phần cứng của các chương trình đồ họa ba chiều phần nào đã được giải quyết, chính vì vậy các công nghệ về đồ họa ba chiều đang rất được quan tâm và phát triển trên thế giới. Các nhóm chương trình ứng dụng của đồ họa ba chiều có thể được kể ra như :

- Hỗ trợ thiết kế : Một trong những ứng dụng của đồ họa ba chiều trên máy tính là các chương trình hỗ trợ thiết kế như CAD, 3D Max, Maya, Poser, … Các chương trình này được sử dụng cho các công việc như thiết kế nhà cửa, quần áo, phương tiện giao thông, các dụng cụ, các mô hình và cả con người, …
- Giáo dục và đào tạo : Các chương trình mô phỏng (thực tại ảo) : mô phỏng sinh họa, hóa học, vật lý học, mô phỏng phóng tàu vũ trụ, lái xe, lái máy bay, các bảng đồ thông tin địa lý GIS…

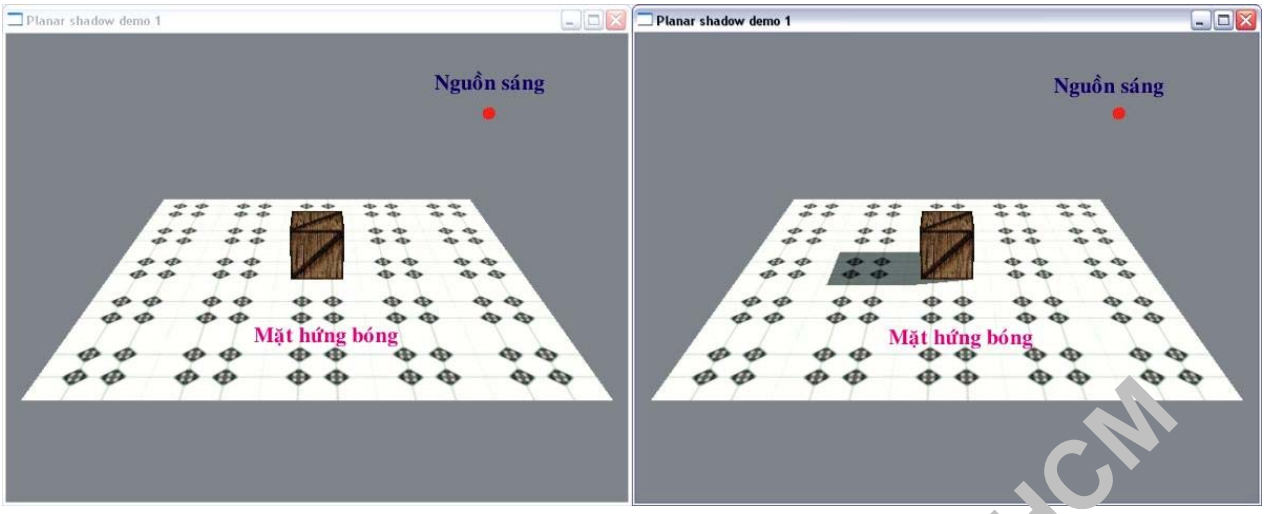
- Giải trí và nghệ thuật : Các chương trình thiết kế mỹ thuật, tạo mô hình cho việc quy hoạch,… cho phép tạo dựng và hiệu chỉnh kiến trúc của các công trình, cho phép quan sát ở nhiều góc độ để có một cái nhìn tổng quan về công trình từ đó đưa ra các chỉnh sửa phù hợp. Ngoài ra đồ họa ba chiều còn giúp tạo ra các chương trình trò chơi giải trí; hỗ trợ các kỹ xảo điện ảnh
- Du lịch ảo : cho du khách có thể tham quan và tương tác với các thế giới giống như thế giới thật mà không cần phải tới tận nơi, và tốn các chi phí cho một chuyến du lịch

Vấn đề quan trọng của đồ họa ba chiều hiện nay là làm thế nào thể hiện được các hình ảnh của thế giới lên màn hình máy tính một cách trung thực nhất

I. BÓNG TRONG ĐỒ HỌA BA CHIỀU TƯƠNG TÁC

Trên thế giới thực con người nhìn được thế giới qua ánh sáng, ánh sáng lại đi đôi với bóng, cứ ở đâu có ánh sáng là ở đó có bóng. Do đó nếu thể hiện được hai yếu tố bóng và ánh sáng thì sẽ làm tăng tính trung thực của các hình ảnh ba chiều trên máy tính.

Khi người dùng giao tiếp với một chương trình đồ họa ba chiều trên máy tính họ thường có cảm giác như các đối tượng đang lơ lửng trong không gian trong khi thực tế là chúng đang nằm trên một bề mặt nào đó, do đó nếu camera chỉ đứng yên một chỗ, người dùng rất khó nhận ra được chiều sâu cũng như vị trí tương đối của các đối tượng. Nếu chương trình đồ họa ba chiều đó có cài đặt kỹ thuật làm bóng cho các đối tượng thì sẽ giúp người dùng dễ dàng cảm nhận đúng hơn về thế giới trong chương trình đó. Chính vì vậy, trong các chương trình đồ họa ba chiều, cho dù tạo bóng cho các đối tượng rất dở vẫn hơn là không tạo bóng cho chúng



A **B**

Chúng ta sẽ dễ nhận biết được vị trí của khối hộp tại hình B hơn là hình A

Trong thực tế, ánh sáng là các hạt photon năng lượng, khi ánh sáng chiếu vào một vật thể nào đó, các hạt photon một phần sẽ được hấp thụ, một phần sẽ được phản xạ lại bởi vật thể đó. Nơi nào không có các hạt photon chiếu tới hay được chiếu tới nhưng với cường độ yếu là nơi có bóng. Trong đồ họa ba chiều, cách tính toán để tạo bóng này sẽ tạo ra được bóng như ở ngoài thế giới thực tuy nhiên kỹ thuật này rất phức tạp và tốn nhiều chi phí. Chúng ta sẽ phải giải quyết nhiều bài toán khó trong cách tiếp cận này, đó là bài toán phản xạ, hấp thụ, bài toán giao giữa một tia và một đa giác,… và quan trọng hơn là chúng ta sẽ phải tính toán trên từng điểm ảnh trong không gian của thế giới ba chiều, nếu không gian là một thế giới rộng lớn thì tốc độ xử lý sẽ rất chậm, và vì hệ thống phần cứng máy tính thông dụng hiện này còn hạn chế nên kỹ thuật này không thể thực hiện được trong các hệ thống thời gian thực.

Các kỹ thuật tạo bóng thực thường được áp dụng trên các thiết bị chuyên dụng đặc biệt rồi xuất ra thành phim hay dựng lên các mô hình ba chiều tĩnh. Có hai kỹ thuật tạo bóng thực nổi tiếng là Ray Trace và Radiosity tuy nhiên luận văn này không chú trọng tới việc xây dựng các thuật toán tạo

bóng này mà tập trung vào nghiên cứu các thuật toán tạo bóng trong các hệ thống mang tính thời gian thực

Trong các hệ thống thời gian thực, chúng ta phải áp dụng các thuật toán tạo bóng giả, các thuật toán này không có độ chính xác 100% tuy nhiên chúng cho ra bóng có thể chấp nhận được đối với thị giác con người. Độ phức tạp cũng như chi phí tính toán của các thuật toán này thấp hơn nhiều so với các thuật toán tạo bóng thực do đó các thuật toán này rất khả thi trên các hệ thống máy tính thông thường và áp dụng được trong các chương trình đồ họa ba chiều mang tính tương tác, thời gian thực. Từ phần này của luận văn khi nói về thuật toán tạo bóng, xin hiểu là thuật toán tạo bóng giả

Các thuật toán tạo bóng (giả) trong đồ họa ba chiều đã được nghiên cứu từ những năm 80 của thế kỷ trước. Có thể kể ra những thành tựu lớn về việc nghiên cứu bóng trong thời gian này, đó là :

- Thuật toán về tạo bóng dựa trên phép chiếu của Jim Blinn năm 1988 và cải tiến của nó vào năm 1996
- Thuật toán tạo bóng trên mặt phẳng của Thant Tessman năm 1989
- Ý tưởng sơ khai về shadow volume của Pllippe Bergeron năm 1986
- Ý tưởng về việc sử dụng thành phần độ sâu của William Reeves, David Salesin và Robert Cook năm 1987

Tuy nhiên đây chỉ là các ý tưởng thuật toán sơ khai, chưa hoàn chỉnh và thiếu sự hỗ trợ về mặt công nghệ nên chưa thể thực hiện tốt được trong thời gian đó và phải tới giữa thập kỷ 1990 các vấn đề về việc tạo bóng trong đồ họa ba chiều tương tác mới có thể thực hiện đúng theo các thuật toán được trên máy tính

Từ giữa thập kỷ 1990 tới nay là thời gian có nhiều thành tựu to lớn nhất về bóng với :

- Thuật toán tạo bóng trên các mặt phẳng (Planar Shadow) của Schilling, Andreas, G.Knittel và Wolfgang năm 1996
- Thuật toán vùng bóng (shadow volume) của Heidmann năm 1991 – và các cải tiến của Tom McReynolds và David Blythe năm 1997 – của Yen Kwoon, Eric Lengyel năm 2002, cải tiến của Tom Hall năm 2003
- Thuật toán Shadow Mapping của Mark J.Kilgard năm 2000
- Thuật toán Projective Shadow Mapping của Mark A.Deloura năm 2000

Mặc dù vậy, vì tính phức tạp khi cài đặt các thuật toán trong các thế giới rộng lớn và các thuật toán đều có những ưu điểm và khuyết điểm, rất khó để áp dụng trong trường hợp tổng quát nên trên thực tế chỉ có một số ít chương trình hiện nay có cài đặt các kỹ thuật tạo bóng.

II. MỤC TIÊU CỦA LUẬN VĂN

Bài toán tạo bóng giả là một bài toán khó trong đồ họa ba chiều thời gian thực, và hiện đang là một trong những đề tài thu hút nhiều sự quan tâm tại các trung tâm nghiên cứu và các trường đại học trên thế giới. Đây cũng là một lĩnh vực chưa được giải quyết trọn vẹn, vẫn còn mở ra nhiều hướng nghiên cứu, phát triển trong tương lai. Trước thực trạng như vậy đề tài này hướng tới 2 mục tiêu như sau

- Nghiên cứu và cài đặt thử nghiệm các thuật toán tạo bóng phổ biến và hiệu quả cao trong đồ họa ba chiều thời gian thực hiện nay, bao gồm :
    - Thuật toán tạo bóng phẳng Planar Shadow
    - Thuật toán tạo bóng Shadow Volume
    - Thuật toán Projective Shadow Mapping

- Xây dựng mô hình nhà tù Chuồng Cọp thuộc Bảo Tàng Di Tích Chiến Tranh tại thành phố Hồ Chí Minh, và cài đặt các thuật toán tạo bóng trên mô hình này nhằm minh họa bóng trong một thế giới tương tác thời gian thực

III. CẤU TRÚC CỦA LUẬN VĂN

Nội dung của luận văn được chia ra làm 4 phần

- **Phần 1** – Phần mở đầu
- **Phần 2** – Kỹ thuật kiểm tra stencil trên từng điểm ảnh : Phần này trình bày về kỹ thuật kiểm tra stencil trên từng điểm ảnh sẽ được áp dụng cho các thuật toán tạo bóng để giới hạn vùng bóng. Đây là phần tùy chọn mô tả một kỹ thuật của đồ họa ba chiều có thể bỏ qua nếu đã biết
- **Phần 3** – Các thuật toán tạo bóng : Giới thiệu về các thuật toán tạo bóng được nghiên cứu trong luận văn, đưa ra đánh giá về ưu và khuyết điểm của mỗi thuật toán
- **Phần 4** – Đánh giá và các hướng phát triển : Đánh giá kết quả đạt được của luận văn và các hướng phát triển

IV. THUẬT NGỮ VÀ CHỮ VIẾT TẮT

Thuật ngữ	Ý nghĩa
Pixel	Một điểm ảnh trong không gian ba chiều, có các thuộc tính chính là màu, độ sâu, độ alpha và giá trị stencil
Texture	Các hình ảnh được dán lên một bề mặt nào đó, làm cho bề mặt giống thật hơn

Render	Thực hiện một quá trình biến đổi để cho ra một hình ảnh ba chiều dạng đầy đủ ra màn hình
Stencil	Một thuộc tính của điểm ảnh trong không gian ba chiều
Stencil Buffer	Bộ đệm chứa các giá trị thuộc tính stencil của điểm ảnh
Stencil Test	Quá trình kiểm tra thuộc tính stencil của từng điểm ảnh, sử dụng các hàm kiểm tra (stencil function) được hỗ trợ bởi phần cứng và phần mềm
Depth Buffer	Bộ đệm chứa các giá trị thuộc tính chiều sâu của điểm ảnh
Depth Test	Quá trình kiểm tra thuộc tính chiều sâu của từng điểm ảnh, sử dụng các hàm kiểm tra (depth function) được hỗ trợ bởi phần cứng và phần mềm
Frame Buffer	Là bộ đệm chứa điểm ảnh đã có đầy đủ các tính chất, chờ được render ra màn hình
Camera	Chỉ tọa độ của điểm quan sát thế giới, ta có thể hình dung đây là điểm đặt camera và chúng ta nhìn thấy được thế giới qua camera này
Volume	Một vùng không gian được tạo ra khi chiếu một vật thể ta xa từ một điểm
View Volume	Khối quan sát, chỉ có các đối tượng nằm trong view volume mới được hiển thị ra màn hình
View Frustum	Là một khối chóp cụt, là kết quả từ phép chiếu

	phối cảnh trên khối quan sát
Mặt phẳng Near	Mặt phẳng đáy gần phía camera của View Frustum
Mặt phẳng Far	Mặt phẳng đáy ở xa phía camera của View Frustum
Field of View	Góc của View Frustum
Silhouette	Hình bao của một vật thể nhìn từ một điểm
Blend	Thuật ngữ chỉ kỹ thuật dùng các toán tử biến đổi hai ngôi trên hai điểm ảnh

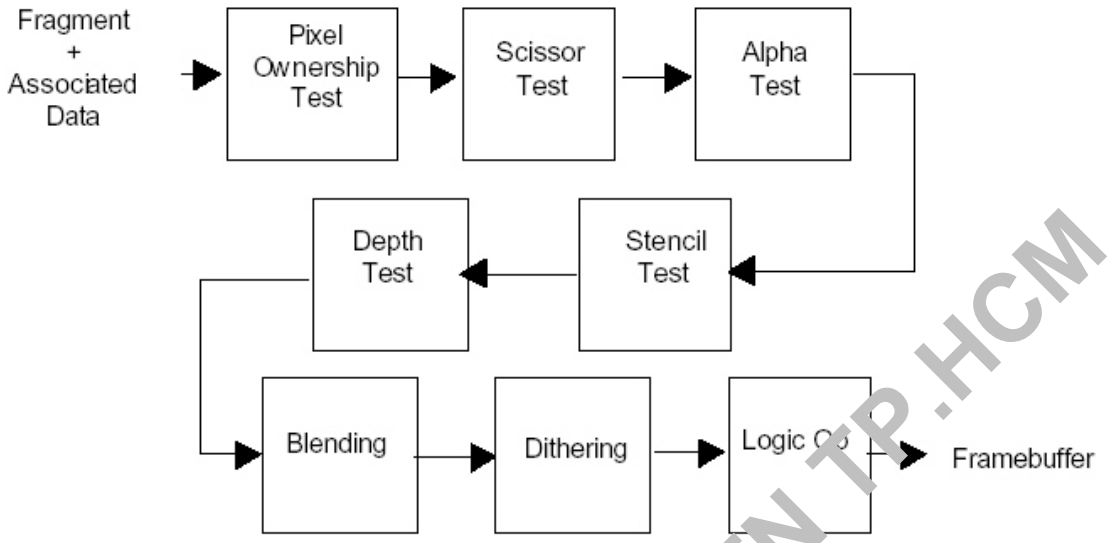
PHẦN 2 : KỸ THUẬT KIỂM TRA STENCIL TRÊN TỪNG ĐIỂM ẢNH

I. GIỚI THIỆU

Nếu chỉ có một nguồn sáng, một vật hứng sáng, một mặt hứng bóng thì các thuật toán tạo bóng rất đơn giản do chỉ có một bóng duy nhất đổ lên một mặt, tuy nhiên khi chúng ta dựng lên một thế giới ba chiều mô phỏng thế giới thực thì thế giới ba chiều đó không chỉ có một mà có rất nhiều nguồn sáng, vật chắn sáng và mặt hứng bóng, do đó có rất nhiều bóng đổ lên nhiều mặt khác nhau và có khi một bóng đổ lên rất nhiều mặt. Vấn đề đặt ra là làm sao chúng ta có thể điều chỉnh bóng sao cho chúng chỉ đổ lên những nơi nào bị chắn sáng bởi các vật thể chắn sáng và với nguồn sáng tương ứng mà thôi. Trong kỹ thuật planar shadow và shadow volume chúng ta sử dụng kỹ thuật kiểm tra stencil trên từng điểm ảnh (stencil test) để giải quyết vấn đề này

Silicon Graphics đã giới thiệu về kỹ thuật kiểm tra stencil trên từng điểm ảnh trong phần cứng cách đây hơn mười năm trước, tuy nhiên vào thời điểm đó giá cho một thiết bị phần cứng hỗ trợ kỹ thuật stencil test rất đắt và chỉ hỗ trợ 4-bit stencil buffer. Hiện nay các card đồ họa có hỗ trợ kỹ thuật stencil test đã xuất hiện rất phổ biến và giá thành thấp, hỗ trợ từ 8 → 64 bit stencil buffer, các card đồ họa này cũng cung cấp một bộ các hàm 3D API hỗ trợ cho kỹ thuật stencil test trên phần cứng. Quá trình dữ liệu

qua các quá trình kiểm tra trước khi được đưa vào framebuffer để render ra màn hình được mô tả như sau :



Trong đó Fragment là một điểm ảnh đã có đầy đủ các tính chất của mình như màu, độ alpha, giá trị stencil, giá trị độ sâu, … . Các quá trình kiểm tra như Sissor Test, Alph Test,Blending, … là các quá trình kiểm tra các thuộc tính của điểm ảnh để có những xử lý tương ứng, chúng ta chỉ quan tâm tới stencil test và depth test. Hai quá trình kiểm tra này chúng tôi sẽ giới thiệu chi tiết hơn ở các phần phía sau

Stencil buffer được hỗ trợ cả trong hai thư viện đồ họa ba chiều rất mạnh và phổ biến hiện nay là OpenGL và DirectX. Trong một chương trình ta có bật tắt chế độ hỗ trợ stencil buffer và điều chỉnh số bit của stencil buffer. Hầu hết các phần mềm, thư viện đều hỗ trợ 8-bit stencil buffer còn phần cứng thì hỗ trợ nhiều loại thông dụng nhất là 4, 8, 16 và 32-bit. Khi thực hiện kỹ thuật stencil trên từng điểm ảnh, nếu phần cứng có hỗ trợ stencil buffer thì phần mềm sẽ tự động chuyển quyền xử lý qua cho phần cứng còn nếu không sẽ dùng phần mềm để thực hiện và tốc độ giảm đi đáng kể

II. Ý TƯỞNG CHÍNH

Kỹ thuật kiểm tra stencil trên từng điểm ảnh (stencil test) là một kỹ thuật thực hiện trên từng điểm ảnh tương tự như kỹ thuật kiểm tra độ sâu của từng điểm ảnh trong đồ họa ba chiều (depth test), depth test kiểm tra và điều chỉnh giá trị độ sâu của điểm ảnh trong depth buffer còn stencil test kiểm tra và điều chỉnh giá trị stencil của điểm ảnh trong stencil bufer. Trong kỹ thuật stencil test khi render, mỗi một điểm ảnh được gán cho một giá trị stencil và giá trị này sẽ được thay đổi và được kiểm tra lại ở lần render kế tiếp, các xử lý tương ứng sẽ được thực hiện đối với từng sự thay đổi.

Trong một thế giới ba chiều có bóng, một điểm ảnh có thể ở một trong hai trạng thái, nằm trong vùng bóng hay nằm ngoài vùng bóng. Chúng ta sẽ gán giá trị stencil tương ứng cho tất cả cả các điểm ảnh, sau đó điều chỉnh các hàm của kỹ thuật stencil test sao cho chỉ cập nhật giá trị stencil của các điểm ảnh nằm trong vùng bóng và render lại thế giới ba chiều này mà không có ánh sáng. Sau đó lại điều chỉnh các hàm của kỹ thuật stencil test sao cho chỉ cập nhật giá trị stencil cuả các điểm ảnh nằm ngoài vùng bóng, render lại thế giới ba chiều một lần nữa với ánh sáng. Các điểm ảnh nằm trong vùng bóng sẽ được render trong trạng thái không có ánh sáng và các điểm ngoài vùng bóng sẽ được render trong trạng thái có ánh sáng và tạo ra bóng trong thế giới ba chiều. Vấn đề đặt ra là làm sao điều chỉnh giá trị stencil của các điểm ảnh nằm trong hay ngoài vùng bóng?

III. KỸ THUẬT KIỂM TRA STENCIL TRÊN TỪNG ĐIỂM ẢNH

Một điểm ảnh thông thường sẽ có giá trị màu (lưu trong color buffer) và giá trị độ sâu (lưu trong depth buffer), khi được render ra màn hình, điểm ảnh sẽ có giá trị màu tương ứng trong khi giá trị độ sâu không được thể

hiện tường minh mà chỉ để quyết định điểm nào nằm trước điểm nào nằm sau, điểm nào được nhìn thấy, điểm nào không từ đó chương trình sẽ quyết định có render điểm ảnh đó ra màn hình hay không. Cũng giống như depth buffer, stencil buffer là tập hợp các giá trị không được vẽ ra màn hình của một điểm ảnh, stencil buffer chứa các giá trị stencil dùng để bổ sung thêm thông tin cho các giá trị của điểm ảnh

Giá trị stencil là một số nguyên không dấu, các thao tác tăng, giảm, so sánh và đánh dấu bit có thể thực hiện được trên các giá trị nguyên này. Khi bắt đầu render một thế giới ba chiều sử dụng kỹ thuật stencil test và stencil buffer phải thiết lập tất cả các giá trị stencil của các điểm ảnh trong stencil buffer về một giá trị nào đó, sau đó sẽ bật hay tắt quá trình kiểm tra giá trị stencil cho từng điểm ảnh, nếu kết quả kiểm tra sai không thực hiện quá trình xác lập các thuộc tính mong muốn nào đó cho một điểm ảnh trong frame buffer để render ra màn hình, nếu kết quả kiểm tra đúng hay chế độ kiểm tra stencil không được bật lên thì thực hiện quá trình xác lập các thuộc tính mong muốn nào đó cho một điểm ảnh trong frame buffer để render ra màn hình. Quá trình kiểm tra stencil sẽ so sánh giá trị stencil trong stencil buffer với giá trị stencil đang tham chiếu tới. Để minh họa các hàm trong kỹ thuật này, chúng tôi xin mượn các hàm trong thư viện đồ họa OpenGL để đưa ra một ví dụ thực tiễn

Trước khi render một thế giới, stencil buffer phải được xóa về một giá trị nào đó đã được định sẵn, trong OpenGL công đoạn này được thực hiện như sau :

```
glClearStencil(0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
GL_STENCIL_BUFFER_BIT);
```

Cũng như kỹ thuật depth test, kỹ thuật stencil test cũng có thể được bật lên hay tắt đi. Khi được bật lên kỹ thuật stencil test sẽ được thực hiện trên từng điểm ảnh. Chúng ta bật tắt kỹ thuật này như sau :

```
glEnable(GL_STENCIL_TEST);
glDisable(GL_STENCIL_TEST);
```

Có 8 hàm so sánh : NEVER, ALWAYS, LESS THAN, LESS THAN OR EQUAL, GREATER THAN, GREATER THAN OR EQUAL, EQUAL và NOT EQUAL. Để thực hiện nhanh hơn so với việc so sánh trực tiếp, trước khi so sánh cả giá trị stencil trong stencil buffer lẫn giá trị stencil đang được tham chiếu tới đều được so sánh từng bit với một mặt nạ bit so sánh stencil. Trong OpenGL hàm so sánh giá trị stencil, giá trị stencil tham chiếu, mặt nạ bít so sánh các giá trị stencil được thể hiện như sau :

```
glStencilFunc(GL_EQUAL, // hàm so sánh
              0x1, // giá trị tham chiếu
              0xff); // mặt nạ so sánh
```

Vì sau khi quá trình kiểm tra độ sâu các giá trị trong depth buffer sẽ được cập nhật để quyết định điểm nào được render ra màn hình và chúng ta chỉ quan tâm đến điểm nào được render ra màn hình thôi nên quá trình kiểm tra stencil phụ thuộc vào quá trình kiểm tra độ sâu do đó kết quả của quá trình kiểm tra stencil lại được chia nhỏ ra thành 3 loại :

- Kiểm tra stencil sai
- Kiểm tra stencil đúng + kiểm tra độ sâu sai
- Kiểm tra stencil đúng + kiểm tra độ sâu đúng

Ứng với từng kết quả chúng ta sẽ thực hiện các thao tác tương ứng cho các giá trị stencil trong buffer.

Có 6 thao tác thay đổi giá trị trong stencil buffer :

- keep : không thay đổi giá trị stencil của điểm ảnh trong stencil buffer

- replace : thay đổi giá trị stencil của điểm ảnh trong stencil buffer với giá trị tham chiếu
- zero : xóa giá trị stencil của điểm ảnh trong stencil buffer về 0
- increment : tăng thêm một đơn vị vào giá trị stencil của điểm ảnh trong stencil buffer cho tới sao cho vẫn nhỏ hơn giá trị lớn nhất
- decrement : giảm một đơn vị vào giá trị stencil của điểm ảnh trong stencil buffer cho tới sao cho vẫn lớn hơn 0
- invert : đảo ngược bit của giá trị stencil của điểm ảnh trong stencil buffer

Một hàm thực hiện các thao tác stencil trong OpenGL có dạng như sau :

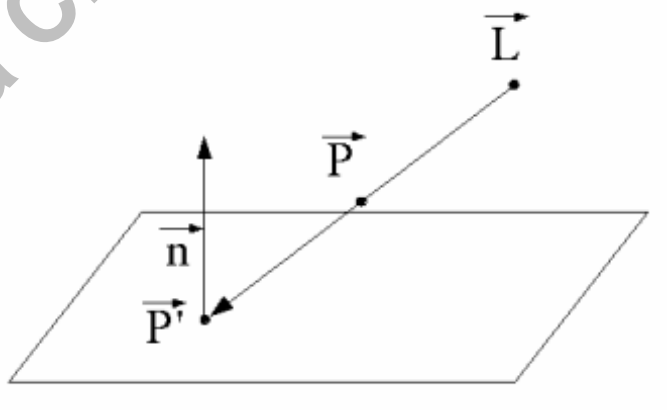
```
glStencilOp(GL_KEEP, // kiểm tra stencil sai
GL_DECR, // kiểm tra stencil đúng, kiểm tra độ sâu sai
GL_INCR); // kiểm tra stencil đúng, kiểm tra độ sâu đúng
glStencilMask(0xff);
```

PHẦN 3 : CÁC THUẬT TOÁN TẠO BÓNG

I. PLANAR SHADOW

1. Giới thiệu

Trường hợp đơn giản nhất của bóng là khi một vật thể đổ bóng trên một mặt phẳng, lúc đó ta có thể chiếu vật thể đó lên mặt phẳng và rút ra được ma trận chiếu ứng với nguồn sáng và mặt phẳng, hình chiếu này chính là hình dạng của bóng của vật thể trên mặt phẳng ; đó chính là ý tưởng chính của thuật toán tạo bóng planar



Nguồn sáng L chiếu một điểm P xuống mặt phẳng thành P'

Giả sử nguồn sáng L đổ bóng một vật thể xuống mặt phẳng D, gọi P(x,y,z,1) là một điểm bất kỳ trên vật thể, hình chiếu của nó trên mặt phẳng là P'(x',y',z',1). Gọi E là một điểm thuộc mặt phẳng hứng bóng và n là vector pháp tuyến của mặt phẳng. Ta có :

- Đường thẳng đi qua nguồn sáng và điểm P là :

$$\vec{x} = \vec{L} + \lambda(\vec{P} - \vec{L})$$

- Mặt phẳng hứng bóng (dạng Euler)

$$(\vec{x} - \vec{E})\vec{n} = 0$$

- Đường thẳng này cắt mặt phẳng tại P', thế x trong đường thẳng vào mặt phẳng ta được :

$$(\vec{L} + \lambda(\vec{P} - \vec{L}) - \vec{E})\vec{n} = 0$$

- Từ đó suy ra

$$\lambda = \frac{\vec{E}\vec{n} - \vec{L}\vec{n}}{\vec{n}(\vec{P} - \vec{L})}$$

- Thế lại vào phương trình đường thẳng ta được điểm P' tính như sau :

$$\vec{P'} = \vec{L} + \frac{\vec{E}\vec{n} - \vec{L}\vec{n}}{\vec{n}(\vec{P} - \vec{L})}(\vec{P} - \vec{L})$$

- Ta đã có tọa độ của một điểm chiếu từ một điểm trên vật chắn sáng, vấn đề là làm sao đưa ra ma trận chiếu M, để đơn giản hóa các hệ số của ma trận ta gán

$$d = \vec{L}\vec{n} \quad \text{và} \quad c = \vec{E}\vec{n} - d$$

- P’ được tính lại như sau

$$\vec{P'} = \vec{L} + \frac{c}{\vec{n}(\vec{P} - \vec{L})}(\vec{P} - \vec{L})$$

và

$$\vec{P'} = \vec{L} + \frac{c}{\vec{n}\vec{P} - d}(\vec{P} - \vec{L})$$

- Từ đó ta có thể viết lại ba thành phần của vector P’ để tính toán ra ma trận chiếu

$$\vec{P'} = \begin{pmatrix} L_x + \frac{c(P_x - L_x)}{\vec{n}\vec{P} - d} \\ L_y + \frac{c(P_y - L_y)}{\vec{n}\vec{P} - d} \\ L_z + \frac{c(P_z - L_z)}{\vec{n}\vec{P} - d} \end{pmatrix}$$

- Khai triển tích vô hướng nP

$$\vec{n}\vec{P} = n_x P_x + n_y P_y + n_z P_z$$

ta được

$$\vec{P'} = \begin{pmatrix} L_x + \frac{c(P_x - L_x)}{n_x P_x + n_y P_y + n_z P_z - d} \\ L_y + \frac{c(P_y - L_y)}{n_x P_x + n_y P_y + n_z P_z - d} \\ L_z + \frac{c(P_z - L_z)}{n_x P_x + n_y P_y + n_z P_z - d} \end{pmatrix}$$

- Quy đồng mẫu số ở mỗi thành phần

$$\vec{P'} = \begin{pmatrix} \frac{L_x(n_xP_x+n_yP_y+n_zP_z-d)+c(P_x-L_x)}{n_xP_x+n_yP_y+n_zP_z-d} \\ \frac{L_y(n_xP_x+n_yP_y+n_zP_z-d)+c(P_y-L_y)}{n_xP_x+n_yP_y+n_zP_z-d} \\ \frac{L_z(n_xP_x+n_yP_y+n_zP_z-d)+c(P_z-L_z)}{n_xP_x+n_yP_y+n_zP_z-d} \end{pmatrix}$$

- Khai triển ra ta được

$$\vec{P'} = \begin{pmatrix} \frac{L_xn_xP_x+L_xn_yP_y+L_xn_zP_z-dL_x+c(P_x-L_x)}{n_xP_x+n_yP_y+n_zP_z-d} \\ \frac{L_yn_xP_x+L_yn_yP_y+L_yn_zP_z-dL_y+c(P_y-L_y)}{n_xP_x+n_yP_y+n_zP_z-d} \\ \frac{L_zn_xP_x+L_zn_yP_y+L_zn_zP_z-dL_z+c(P_z-L_z)}{n_xP_x+n_yP_y+n_zP_z-d} \end{pmatrix}$$

- Gom nhóm theo Px, Py và Pz ta được

$$\vec{P'} = \begin{pmatrix} \frac{P_x(L_xn_x+c)+P_y(L_xn_y)+P_z(L_xn_z)-cL_x-dL_x}{n_xP_x+n_yP_y+n_zp_z-d} \\ \frac{P_x(L_yn_x)+P_y(L_yn_y+c)+P_z(L_yn_z)-cL_y-dL_y}{n_xP_x+n_yP_y+n_zp_z-d} \\ \frac{P_x(L_zn_x)+P_y(L_zn_y)+P_z(L_zn_z+c)-cL_z-dL_z}{n_xP_x+n_yP_y+n_zp_z-d} \end{pmatrix}$$

Gọi ma trận chiếu là M và có các hệ số là m11 → m44 ta có tích của một điểm trên vật chắn sáng với ma trận chiếu sẽ là điểm chiếu của nó trên mặt phẳng, hay M*P = P'. Nhân M với P(x,y,z,1) ta được

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xm_{11}+ym_{12}+zm_{13}+m_{14} \\ xm_{21}+ym_{22}+zm_{23}+m_{24} \\ xm_{31}+ym_{32}+zm_{33}+m_{34} \\ xm_{41}+ym_{42}+zm_{43}+m_{44} \end{pmatrix}$$

Kết quả thu được chính là điểm P' nên

$$\frac{P_x m_{11} + P_y m_{12} + P_z m_{13} + m_{14}}{P_x m_{41} + P_y m_{42} + P_z m_{43} + m_{44}} = \frac{P_x(L_x n_x + c) + P_y(L_x n_y) + P_z(L_x n_z) - cL_x - dL_x}{n_x P_x + n_y P_y + n_z p_z - d}$$

$$\wedge \frac{P_x m_{21} + P_y m_{22} + P_z m_{23} + m_{24}}{P_x m_{41} + P_y m_{42} + P_z m_{43} + m_{44}} = \frac{P_x(L_y n_x) + P_y(L_y n_y + c) + P_z(L_y n_z) - cL_y - dL_y}{n_x P_x + n_y P_y + n_z p_z - d}$$

$$\wedge \frac{P_x m_{31} + P_y m_{32} + P_z m_{33} + m_{34}}{P_x m_{41} + P_y m_{42} + P_z m_{43} + m_{44}} = \frac{P_x(L_z n_x) + P_y(L_z n_y) + P_z(L_z n_z + c) - cL_z - dL_z}{n_x P_x + n_y P_y + n_z p_z - d}$$

Từ đây ta có thể dễ dàng rút ra ma trận chiếu M như sau

$$\begin{pmatrix} L_x n_x + c & L_x n_y & L_x n_z & -cL_x - dL_x \\ L_y n_x & L_y n_y + c & L_y n_z & -cL_y - dL_y \\ L_z n_x & L_z n_y & L_z n_z + c & -cL_z - dL_z \\ n_x & n_y & n_z & -d \end{pmatrix}$$

2. Không áp dụng kỹ thuật stencil

Để tạo bóng cho một vật từ một nguồn sáng xuống một mặt phẳng ta chỉ cần tính toán ma trận chiếu từ các thông số của mặt phẳng và nguồn sáng và sau đó chuyển đổi vật thể này bằng ma trận chiếu sẽ được hình hình chiếu của nó trên mặt phẳng sau đó render hình chiếu này này với màu đen và không có ánh sáng ta sẽ được bóng của vật thể xuống mặt phẳng

Thuật toán được phát biểu như sau :

Đầu vào :

- Một nguồn sáng điểm
- Một mặt phẳng
- Một vật chắn sáng

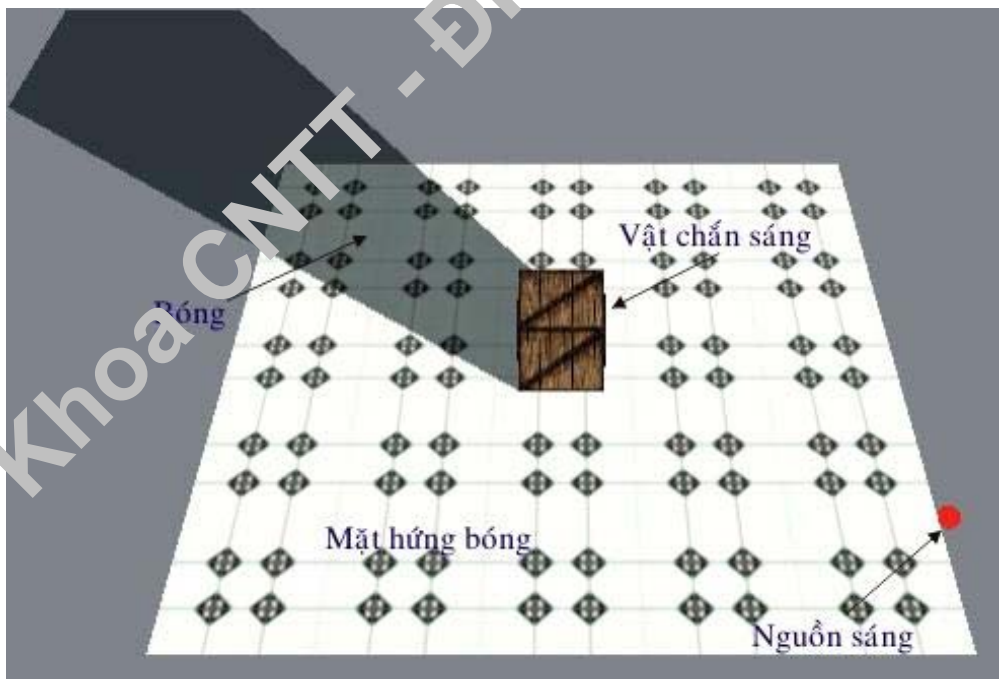
Thuật toán :

- Tính toán ma trận Model View trong phép biến đổi view tranform

- Tính toán phương trình mặt phẳng, bật chế độ ánh sáng lên và render mặt phẳng hứng bóng, vật chắn sáng
- Lưu ma trận Model View
- Tính toán ra ma trận chiếu từ phương trình mặt phẳng và nguồn sáng, sau đó nhân ma trận Model View với ma trận này
  - Render lại vật chắn sáng (lúc này đã được chuyển đổi qua ma trận chiếu) với màu đen, trong chế độ không có ánh sáng
- Phục hồi ma trận Model View

3. Áp dụng kỹ thuật stencil

Phương trình mặt phẳng biểu điễn cho một mặt phẳng vô hạn trong khi đó mặt phẳng hứng bóng lại là hữu hạn nên đôi khi bóng đổ ra ngoài mặt phẳng hứng bóng, đòi hỏi phải sử dụng kỹ thuật stencil test để giải quyết



Bóng đổ ra ngoài do chiếu xuống mặt phẳng trên lý thuyết là vô hạn trong khi mặt hứng bóng trong thực tế lại hữu hạn

Đầu tiên ta render mặt phẳng hứng bóng vào stencil buffer với giá trị stencil là 1 hay nói cách khác các điểm ảnh nằm trong mặt phẳng hứng bóng sẽ có giá trị stencil là 1, sau đó chúng ta render bóng với điều kiện là chỉ render bóng tại điểm ảnh nào có giá trị stencil là 1. Bóng được tạo ra sẽ chỉ nằm trong vùng giới hạn bởi mặt phẳng hứng bóng mà không được render ở các vùng nằm ngoài mặt phẳng hứng bóng

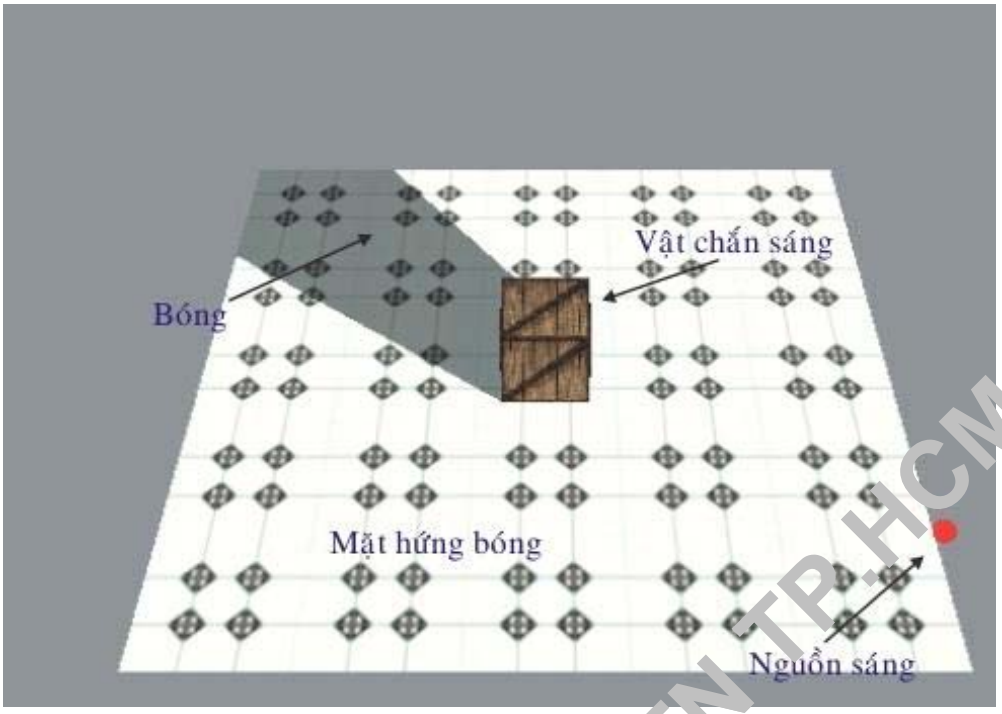
Thuật toán phát biểu như sau **:**

Đầu vào :

- Một nguồn sáng điểm
- Một mặt phẳng
- Một vật chắn sáng

Thuật toán :

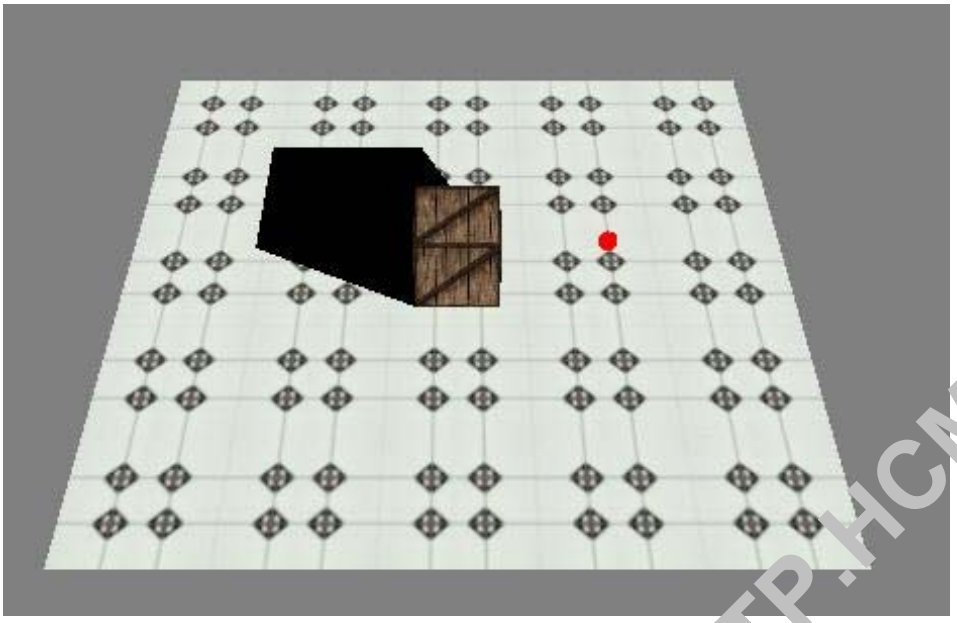
- Tính toán ma trận Model View trong phép biến đổi view tranform
- Bật chế độ kiểm tra stencil, đặt hàm kiểm tra giá trị stencil là luôn gán giá trị stencil = 1
- Tính toán phương trình mặt phẳng, bật chế độ ánh sáng lên và render mặt phẳng hứng bóng, vật chắn sáng
- Lưu ma trận Model View
- Tính toán ra ma trận chiếu từ phương trình mặt phẳng và nguồn sáng, sau đó nhân ma trận Model View với ma trận này
- Đặt hàm kiểm tra là chỉ thông qua khi giá trị stencil là 1
- Render lại vật chắn sáng (lúc này đã được chuyển đổi qua ma trận chiếu) với màu đen, trong chế độ không có ánh sáng
- Phục hồi ma trận Model View
- Tắt chế độ kiểm tra stencil



Chỉ render bóng ở nơi nào thuộc về mặt phẳng hứng bóng dùng kỹ thuật stencil test

4. Các cải tiến quan trọng

Màu của bóng là đen và sẽ che mất nền do đó chúng ta phải blend bóng với nền để cho ra bóng thực hơn

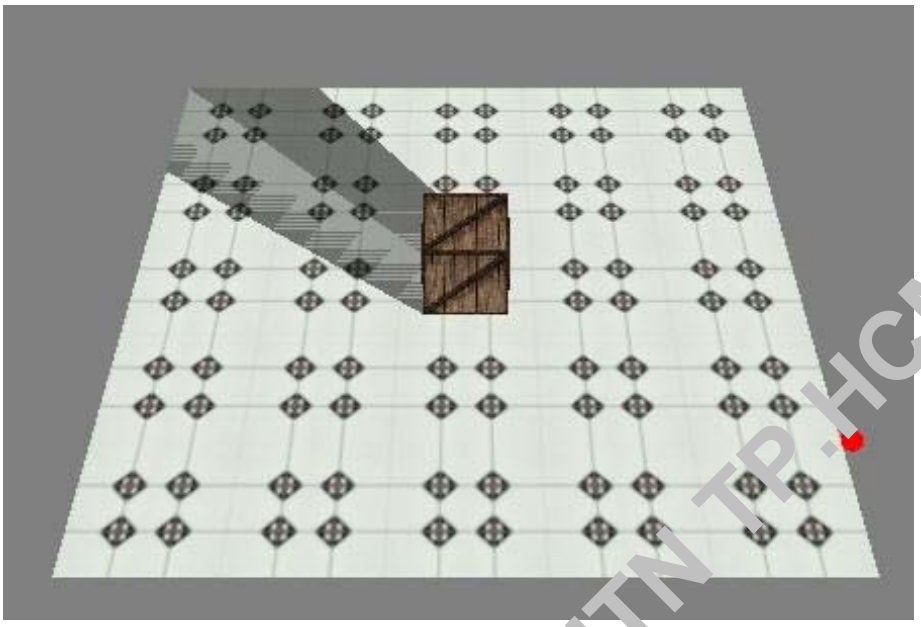


Bóng đổ lên nền gạch trông không thực vì bóng có màu đen và che khuất các chi tiết của nền

Bóng đổ lên đã được blend đi để trông thực hơn

Bóng planar là hình chiếu của vật chắn sáng xuống mặt phẳng, tuy nhiên do việc tính toán và làm tròn số trên số thực nên bóng không hoàn toàn nằm trên mặt phẳng mà có phần của bóng nằm trên mặt phẳng, có phần của bóng nằm dưới mặt phẳng gây ra hiện tượng chập

chờn chỗ thấy chỗ không, để giải quyết vấn đề này chúng ta phải tắt quá trình kiểm tra depth test để cho bóng không bị nền che khuất.



Bóng planar có phần thấy và có phần không thấy do có phần nằm trên và có phần nằm dưới mặt hứng bóng

Ma trận bóng luôn luôn được tính và bóng luôn được render lại ở mỗi một khung hình, cho dù bóng không hề bị thay đổi. Để giải quyết vấn đề này ta render toàn bộ bóng vào một texture mà đã được tạo ra từ trước. Texture bóng này sẽ chỉ được tính lại khi bóng thay đổi (do mặt hứng bóng hay vật chắn sáng hay nguồn sáng thay đổi), texture này chúng ta có thể lưu lại dưới dạng texture có thành phần alpha, nó sẽ được render trên bề mặt của mặt hứng bóng với kỹ thuật blend vì thế những điểm ảnh nằm trong vùng bóng sẽ tối đi và các điểm ảnh nằm ngoài vùng bóng không bị thay đổi

Áp ma trận chiếu vào vật thể chắn sáng đôi khi cũng cho ra bóng không chính xác đó là khi nguồn sáng nằm giữa mặt phẳng hứng bóng và vật thể chắn sáng. Trường hợp này đáng lẽ là không đổ bóng tuy nhiên khi áp ma trận chiếu vào sẽ cho ra bóng đổ ngược để giải quyết vấn đề này

mỗi khi đổ bóng ta phải chọn các mặt phẳng để đổ bóng tương ứng với 1 nguồn sáng và vật chắn sáng, tuy nhiên điều này rất khó thực hiện cho vật thể có số đa giác lớn

5. Ưu điểm

Planar shadow cài đặt khá đơn giản, bóng của vật thể tạo ra có độ chính xác trên mặt phẳng hứng bóng tương ứng với nguồn sáng.

Chi phí tính toán cho bóng không cao nên không làm chậm tốc độ của chương trình, đây là một ưu điểm lớn của planar shadow

Đối với bóng tạo với kỹ thuật stencil, nếu phần cứng có hỗ trợ stencil buffer thì không ảnh hưởng tới tốc độ

6. Khuyết điểm

Chỉ tạo được bóng trên các mặt phẳng

Bóng của vật thể hình chiếu của vật thể nên có số lượng đa giác và độ phức tạp và tốn chi phí render như nhau trong khi bóng trong thực tế đơn giản hơn vật chắn sáng nhiều

Không có khả năng tự đổ bóng lên chính vật thể chắn sáng

7. Nhận xét

Thuật toán tạo bóng Planar Shadow tuy không cài đặt được trong một thế giới có các đối tượng hình học phức tạp và các mặt cong, nhưng rất phù hợp trong các thế giới có dạng hình khối như một căn phòng, toà nhà, hay đổ bóng tĩnh của một vật thể lên sàn nhà, …

II. SHADOW VOLUME

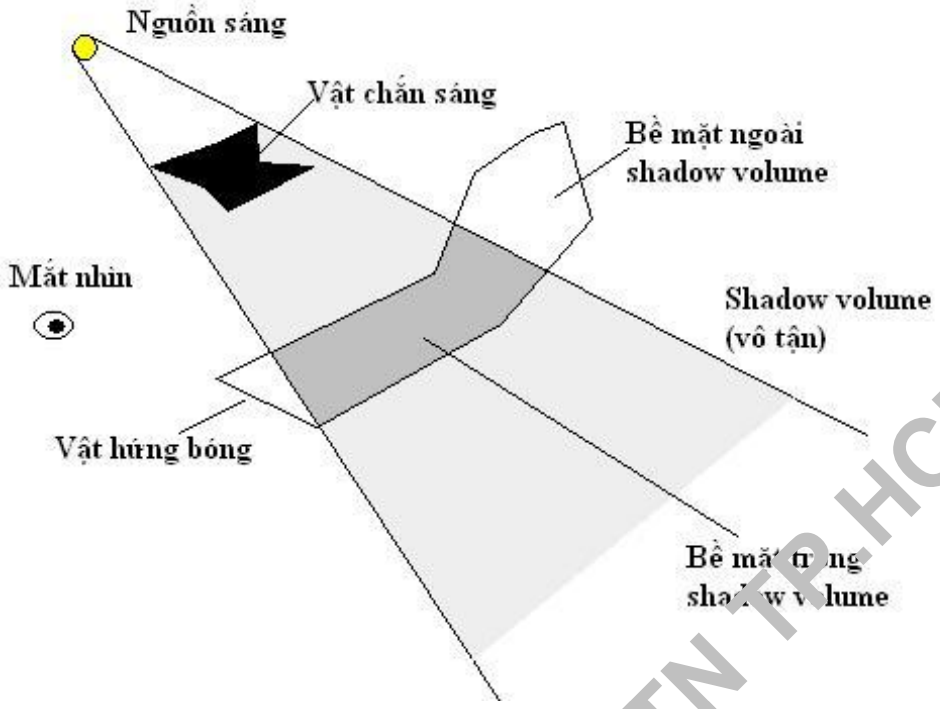
1. Giới thiệu

Trong thực tế nhiều khi bóng đổ lên các bề mặt không phải là mặt phẳng đòi hỏi phải có một kỹ thuật mạnh hơn kỹ thuật planar shadow để tạo ra bóng trong trường hợp này đó là kỹ thuật tạo bóng dựa vào vùng bóng (shadow volume)

Trong thế giới thực khi một vật thể chắn một nguồn sáng nó sẽ tạo ra một vùng không gian không được chiếu sáng bởi nguồn sáng đó, vùng không gian này này được gọi là shadow volume. Ý tưởng chính của thuật toán tạo bóng dựa vào vùng bóng (shadow volume) là coi vùng bóng là một volume

Thuật toán gồm 2 bước

- Tính toán shadow volume được hình thành bởi nguồn sáng và tập các vật chắn sáng
- Xác định rằng một điểm nằm trên bề mặt hứng bóng được chiếu sáng hay bị che bởi một hay nhiều vật thể chắn sáng , để làm được điều này ta phải xác định xem điểm đó nằm trong hay ngoài shadow volume. Nếu điểm đó nằm ngoài shadow volume nó được chiếu sáng bởi nguồn sáng, còn nếu nằm trong shadow volume nó không được chiếu sáng bởi nguồn sáng và tạo ra bóng

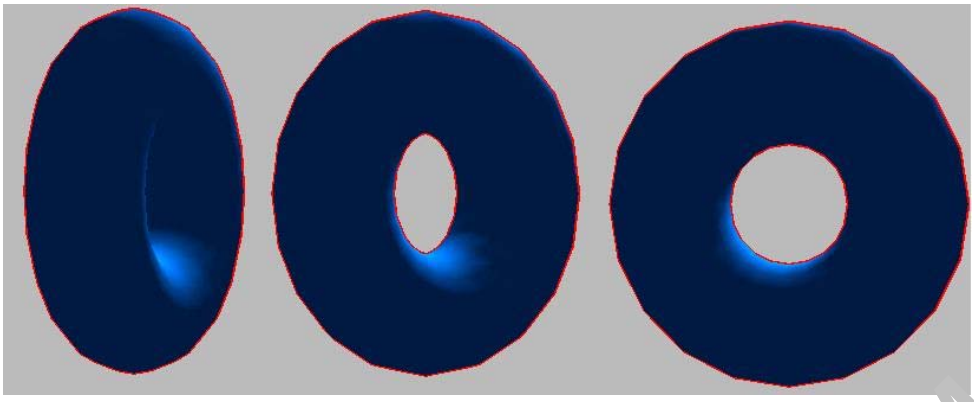


Shadow volume chia các điểm trên bề mặt hứng bóng thành 2 nhóm, nhóm nằm trong shadow volume và nhóm nằm ngoài shadow volume

2. Shadow volume

a. Tính silhouette

Shadow volume tạo bởi một nguồn sáng và một vật thể là một vùng không gian mà ánh sáng bị che bởi vật thể đó, để tạo được shadow volume phải tính được hình bao của vật thể nhìn từ điểm nguồn sáng hình này được gọi là silhouette của vật thể ứng với nguồn sáng, silhouette của vật thể sẽ thay đổi khi nguồn sáng bị thay đổi hay vật thể chuyển động

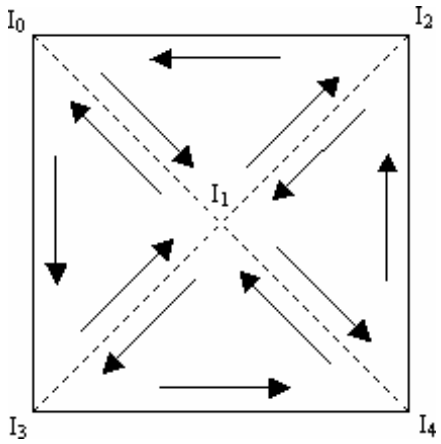


Khi vật chắn sáng thay đổi hay nguồn sáng thay đổi thì silhouette cũng thay đổi theo. Viền đỏ là silhouette của vật thể thay đổi theo từng trường hợp

Thuật toán tìm silhouette của một vật thể đối với một nguồn sáng xác định đòi hỏi vật chắn sáng phải là một lưới khép kín giới hạn gồm các tam giác, điều này có nghĩa là bề mặt của vật chắn sáng phải được chia thành một tập các tam giác. Lưới khép kín nên bất kỳ một cạnh nào cũng được chia sẻ bởi hai tam giác và không có một lỗ hổng nào bên trong vật thể đó, vật chắn sáng phải là một vật thể đặc. Thuật toán tính silhouette là một phần tính toán tốn nhiều chi phí của thuật toán tạo shadow volume, do đó việc tính toán ra silhouette ảnh hưởng rất lớn đối với bóng, nhất là bóng trong thời gian thực (bóng được tính lại trong mọi thời điểm)

Thuật toán của Yen Kwoon, Hun :

Thuật toán này không xem xét hết tất cả các cạnh trên lưới tam giác của vật chắn sáng mà chỉ xem xét các cạnh mà hai tam giác chia sẻ nó có một tam giác hướng tới nguồn sáng và một tam giác không hướng tới nguồn sáng.



Một mặt của khối lập phương được chia ra thành 4 tam giác với thứ tự các đỉnh ngược chiều kim đồng hồ. Các cạnh được render đứt đoạn là các cạnh ta không quan tâm tới, các cạnh render liền nét là các cạnh được xét

Thuật toán được phát biểu như sau :

Đầu vào :

- Vật thể chắn sáng với các mặt đã được chuẩn hóa đưa về dạng lưới các tam giác
- Lưới tam giác của vật thể khép kín
- Một stack lưu các cạnh

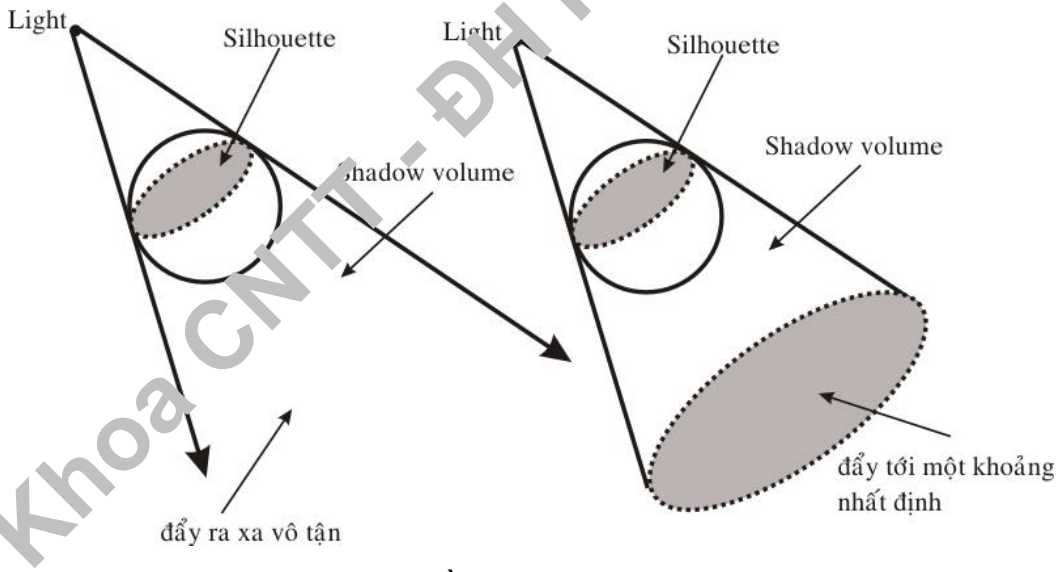
Thuật toán :

- Duyệt qua tất cả các tam giác trong lưới
  - Nếu tam giác đang xét hướng tới nguồn sáng (tích vô hướng vector pháp tuyến của tam giác đó với nguồn sáng > 0)
    - Đẩy ba cạnh (mỗi cạnh gồm hai đỉnh) của nó vào trong một stack cạnh
    - Nếu cạnh đó đã có trong stack (xét cả hai chiều xuôi và ngược – AB & BA) : Xóa cả hai cạnh đó khỏi stack
- Silhouette tìm được là các cạnh trong stack

Chi phí cho việc tính silhouette trong thuật toán này là khá lớn do mỗi lần tính lại silhouette phải kiểm tra tất cả các cạnh của vật chắn sáng, nó làm chậm đáng kể thuật toán tạo bóng shadow volume. Chi phí cho thuật toán tính silhouette này là O(n) với n là số cạnh của vật chắn sáng

b. Tính shadow volume

Sau khi đã tính được silhouette của vật chắn sáng, bước tiếp theo của thuật toán là tính shadow volume của vật chắn sáng đó ứng với nguồn sáng. Để tính shadow volume chúng ta đẩy silhouette của vật thể từ điểm nguồn sáng ra vô tận hay ra một khoảng xác định nào đó, vùng không gian tạo được khi đẩy silhouette gọi là shadow volume của vật chắn sáng



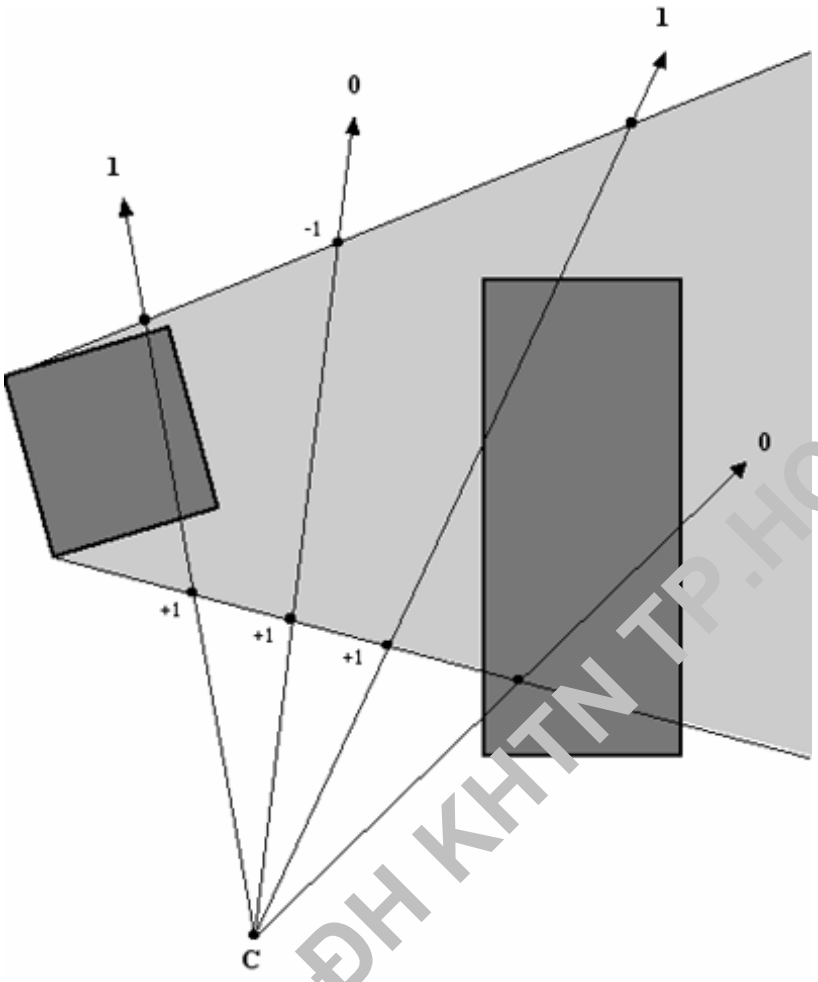
Shadow volume của hình cầu là vô tận hay giới hạn tùy thuộc vào việc đẩy silhouette ra vô tận hay một khoảng xác định

3. Thuật toán shadow volume

a. Thuật toán

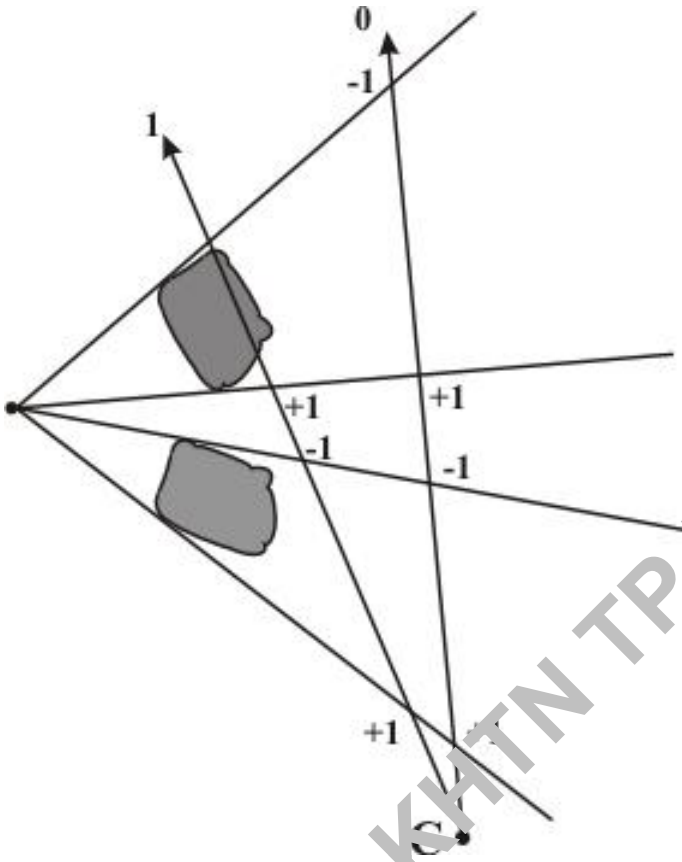
Cho một nguồn sáng điểm đổ bóng một vật chắn sáng lên bề mặt của một vật hướng bóng, shadow volume được tạo ra bởi nguồn sáng và vật chắn sáng sẽ chia không gian ra thành các vùng có bóng và các vùng không có bóng. Ý tưởng chính của thuật toán shadow volume là sử dụng kỹ thuật stencil test để gán các giá trị stencil khác nhau cho các điểm ảnh thuộc vùng bóng và không bóng. Để tạo ra các điểm và các giá trị stencil tương ứng ta sử dụng kết quả của các toán tử trong kỹ thuật stencil test. Chia quá trình render shadow volume làm hai bước :

- Bước 1 : Render mặt trước của shadow volume (gần điểm camera quan sát) sử dụng các toán tử stencil để tăng giá trị stencil trong stencil buffer tại các điểm thông qua quá trình depth test.
- Bước 2 : Render mặt sau của shadow volume, sử dụng các toán tử stencil để giảm giá trị stencil trong stencil buffer tại các điểm thông qua quá trình depth test.

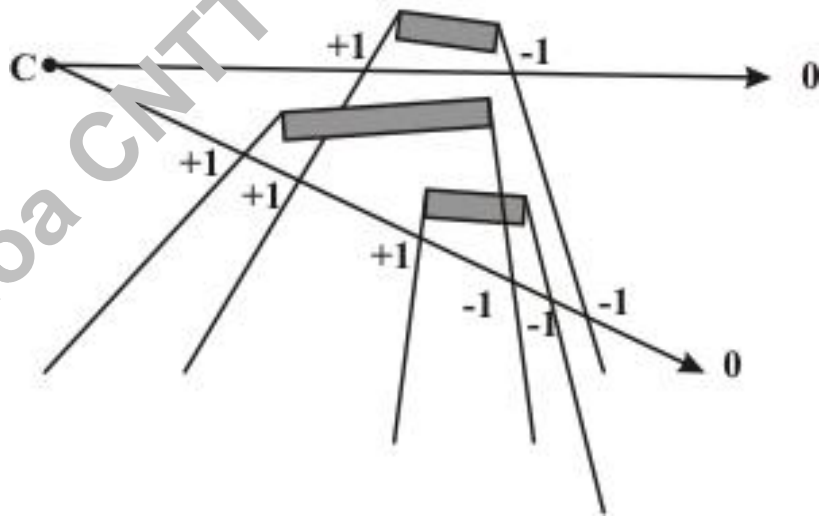


Số ở đầu các mũi tên chỉ ra giá trị stencil được lưu trong stencil buffer. Trường hợp 1 shadow volume

Trong trường hợp có nhiều vật chắn sáng ứng với một nguồn sáng thuật toán sẽ tạo ra nhiều volume và quá trình sẽ trở nên phức tạp hơn. Giá trị stencil sẽ được tính toán, tăng giảm nhiều lần để cho ra kết quả cuối cùng phù hợp với bóng thực sự



Trường hợp 2 volume



Trường hợp nhiều volume

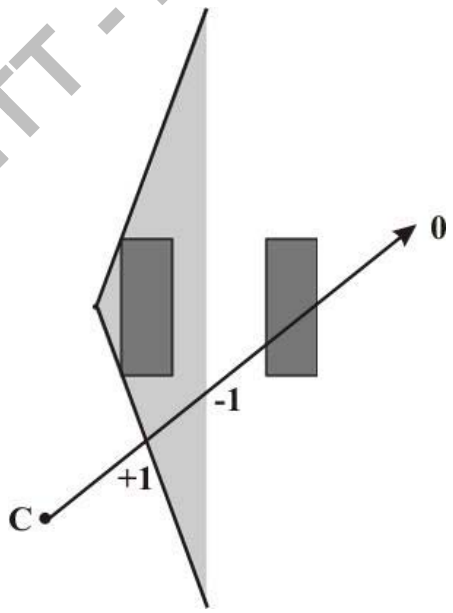
Kỹ thuật này sẽ gán giá trị stencil khác 0 cho tất cả các điểm nằm trong vùng shadow volume, kể cả các điểm thuộc về vật chắn sáng.

Tia đầu tiên bên trái xuất phát từ camera, khi render mặt trước của shadow volume depth test thông qua nên giá trị stencil của điểm tương ứng trên mặt trước của shadow volume là 1, render mặt sau của shadow volume depth test không thông qua nên không thay đổi giá trị stencil của điểm này, ngược lại ở tia thứ hai, cả hai lần depth test đều được thông qua nên giá trị stencil của điểm tương ứng trên

b. Các cải tiến quan trọng

Shadow volume vô tận

Trong thực tế khi cài đặt thuật toán này ta không thể cho shadow volume là vô tận mà chúng ta sẽ cho shadow volume dài một khoảng nào đó tuy nhiên khi vật chắn sáng và nguồn sáng gần nhau thì khoảng giới hạn của shadow volume sẽ ngắn lại, khi đó việc đổ bóng lên vật hứng bóng ở xa sẽ bị sai. Điều này bắt buộc chúng ta phải tìm ra một giải pháp cho việc đẩy shadow volume ra xa vô tận



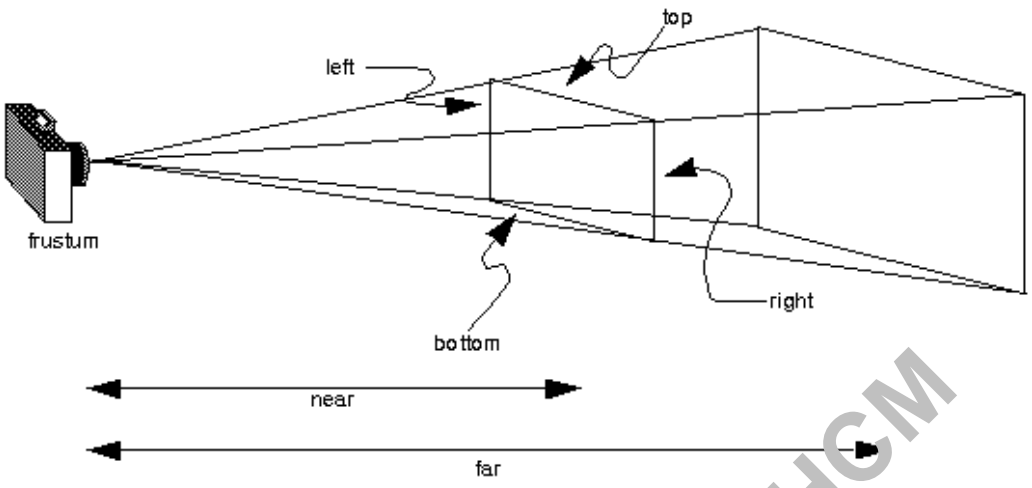
Khi vật chắn sáng gần nguồn sáng, nếu shadow volume là hữu hạn shadow volume sẽ bị co ngắn lại, vật hứng bóng sẽ nằm ngoài shadow volume

Để shadow volume bao bọc được tất cả các vật thể có thể hứng bóng trong thế giới đòi hỏi shadow volume phải dài vô tận nghĩa là ta phải chiếu silhouette của vật thể từ điểm chiếu là nguồn sáng ra xa vô tận. Sử dụng ma trận chiếu chuẩn trong đồ họa ba chiều dẫn tới shadow volume sẽ bị cắt bởi mặt phẳng far của view frustum bởi vì view frustum có kích thước xác định. Để tránh hiện tượng này chúng ta có thể đặt mặt phẳng far của view frustum ra xa vô tận so với nguồn sáng.

Gọi mặt phẳng chiếu ánh xạ một điểm từ hệ tọa độ mắt nhìn sang hệ tọa độ clip trong đồ họa ba chiều là **P.** Khi đang ở hệ tọa độ mắt nhìn, camera đặt ở gốc tọa độ, trục x trỏ sang phải, trục y trỏ lên trên và camera hướng ngược chiều của trục z thì **P** có dạng như sau :

$$\mathbf{P} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

với các hệ số n, f, r, l, t, b tương ứng là các khoảng cách near, far, và các hệ số chỉ tọa độcác đỉnh right, left, top, bottom của view fustum như hình sau :



trong đó (left, bottom, near) chỉ tọa độ (x,y,z) của điểm bên dưới phía trái của mặt phẳng near, …

Khi đặt mặt phẳng far ra xa vô tận ta có ma trận chiếu $\mathbf{P}_\infty$:

$$\mathbf{P}_\infty = \lim_{f\to\infty} \mathbf{P} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Từ đó ta có phép chiếu một điểm từ hệ tọa độ mắt nhìn eye sang hệ tọa độ mới trong đó mặt phẳng far của view frustum ở xa vô tận gọi là hệ tọa độ clip

$$\mathbf{V}_{clip} = \mathbf{P}_\infty \mathbf{V}_{eye} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l}x + \frac{r+l}{r-l}z \\ \frac{2n}{t-b}y + \frac{t+b}{t-b}z \\ -z - 2nw \\ -z \end{bmatrix}$$

Với ma trận chiếu $\mathbf{P}_\infty$, trên lý thuyết mặt phẳng far của view frustum ở xa vô tận, do đó khi chuyển sang hệ tọa độ chuẩn hóa mọi điểm sẽ có thành phần $z < 1$ và nằm trong view frustum. Tuy nhiên vì giới hạn của phần cứng khi tính toán lên số thực đôi khi thành phần z dường như lớn hơn 1 một khoảng lân cận rất nhỏ, giá trị này sau đó chuyển thành số nguyên lưu trong z-buffer do đó quá trình depth test có thể bị sai điều này kéo theo thuật toán shadow volume có thể bị sai bởi vì quá trình render các mặt volume phụ thuộc vào depth test. Để tránh điều này chúng ta có thể ánh xạ tọa độ z của một điểm ở vô tận về một khoảng lân cận dưới của 1, nghĩa là các giá trị của z là D_z nằm trong khoảng [-1, 1] sẽ được ánh xạ sang các giá trị D'_z nằm trong khoảng [-1, 1- ε] với ε là một số rất nhỏ. Ánh xạ này được biểu diễn như sau

$$D'_z = (D_z + 1)\frac{2-\varepsilon}{2} - 1 .$$

Do đó ta phải tính lại ma trận chiếu để cho tọa độ z của hệ tọa độ clip nằm trong khoảng [-1, 1 - ε]. Ta có $D_z = (V_{clip})_z / (V_{clip})_w$ và $D'_z = (V'_{clip})_z / (V_{clip})_w$ do đó

$$\frac{(\mathbf{V}'_{clip})_z}{(\mathbf{V}_{clip})_w} = \left(\frac{(\mathbf{V}_{clip})_z}{(\mathbf{V}_{clip})_w} + 1\right)\frac{2-\varepsilon}{2} - 1$$

Mà trong phép chuyển đổi qua ma trận $\mathbf{P}_\infty$ ta có thành phần $(V_{clip})_w$ là –z nên

$$\frac{(\mathbf{V}'_{\text{clip}})_z}{-z} = \left(\frac{-z-2nv}{-z}+1\right)\frac{2-\varepsilon}{2}-1.$$

Khai triển qua các phép biến đổi đơn giản ta có

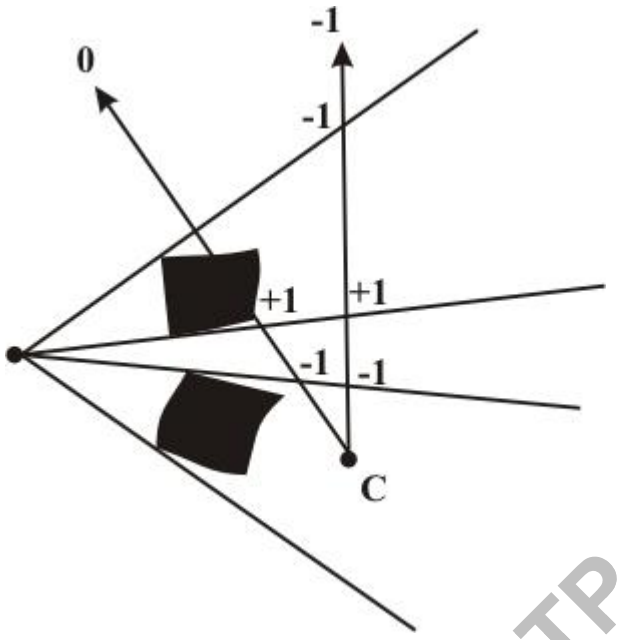
$$(\mathbf{V}'_{\text{clip}})_z = z(\varepsilon-1)+nv(\varepsilon-2).$$

Từ đó ta có thể suy ra ma trận chiếu mới $\mathbf{P'}_\infty$ bằng cách kết hợp giữa ma trận $\mathbf{P}_\infty$ và biểu thức trên.

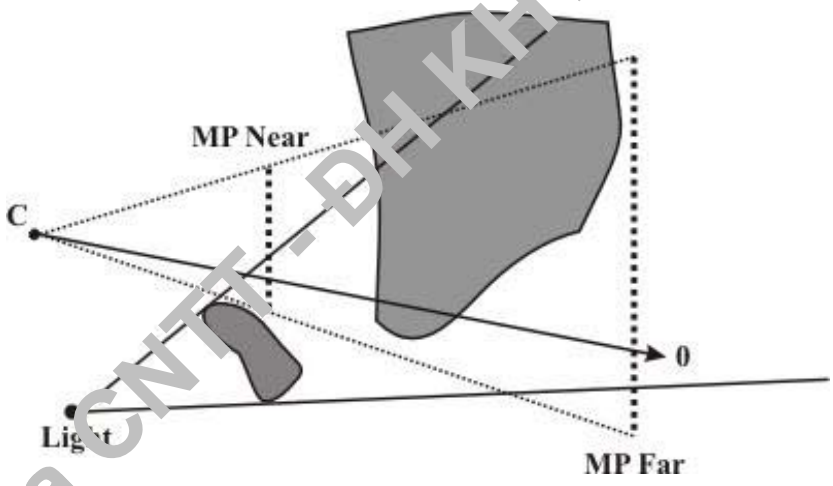
$$\mathbf{P}'_\infty = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \varepsilon-1 & n(\varepsilon-2) \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Khi camera nằm trong shadow volume hay bị cắt bởi mặt phẳng near

Khi camera nằm trong shadow volume hoặc shadow volume bị cắt bởi mặt phẳng near của view frustum. Cả hai trường hợp đều dẫn tới việc đặt các giá trị không chính xác trong stencil buffer do thuật toán shadow volume tổng quát chỉ xem xét trường hợp camera nằm ngoài shadow volume rồi đưa ra phương pháp đặt giá trị stencil vào trong buffer một cách tương ứng



Camera nằm trong shadow volume, theo thuật toán tổmg quát giá trị stencil trong buffer sẽ bị thiết lập sai

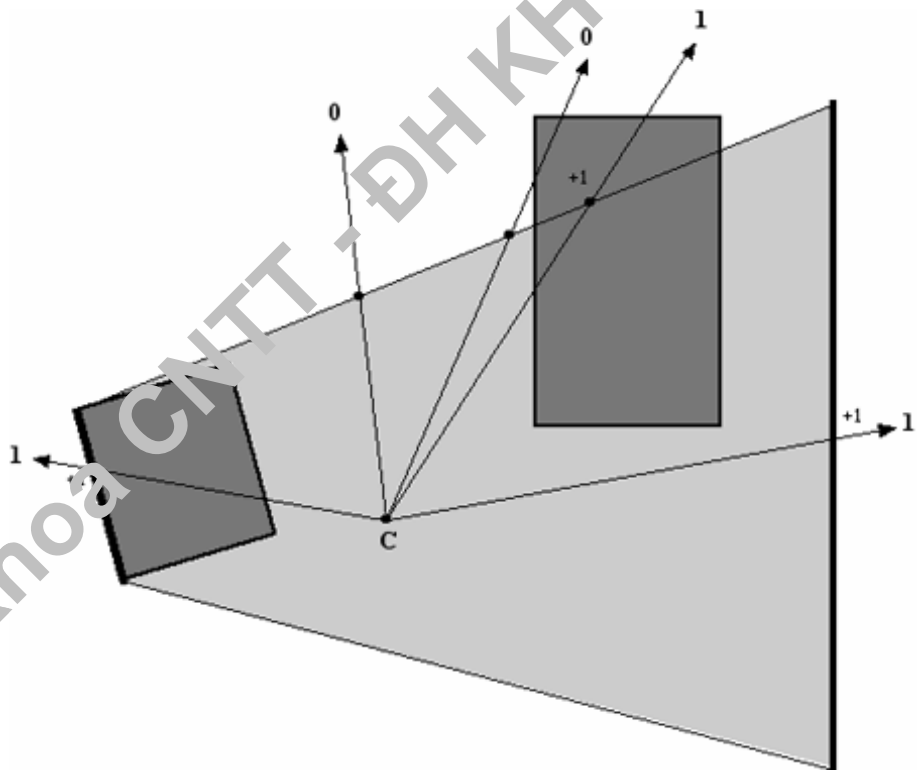


Shadow volume bị cắt bởi mặt phẳng near , theo thuật toán tổmg quát giá trị stencil trong buffer sẽ bị thiết lập sai

Cách giải quyết cho vấn đề này là xác định xem camera có nằm trong shadow volume hay không, để làm được điều này shadow volume phải là một vùng không gian khép kín, Jonh Carmack đề nghị thêm hai nắp đậy cho khoảng không gian của shadow volume để nó trở thành một vùng không gian khép kín cho dù shadow volume là vô tận sau đó sử dụng các toán tử stencil để thiết lập giá

trị stencil tương ứng. Nắp trước của shadow volume được tạo từ các tam giác hướng tới nguồn sáng trong lưới tam giác của vật hứng bóng, nắp sau của shadow volume được tạo bằng cách chiếu các đỉnh của nắp trước ra xa vô tận với tâm chiếu là nguồn sáng. Sau khi đã có một shadow volume khép kín, chúng ta cũng render shadow volume qua 2 bước :

- Render mặt sau của shadow volume sử dụng các toán tử tăng giá trị stencil trong buffer khi quá trình depth test không được thông qua
- Render mặt trước của shadow volume sử dụng các toán tử giảm giá trị stencil trong buffer khi quá trình depth test không được thông qua



Giá trị stencil tương ứng khi camera nằm trong shadow volume

Thuật toán shadow volume được phát biểu như sau :

Đầu vào :

- Các nguồn sáng điểm

Thuật toán :

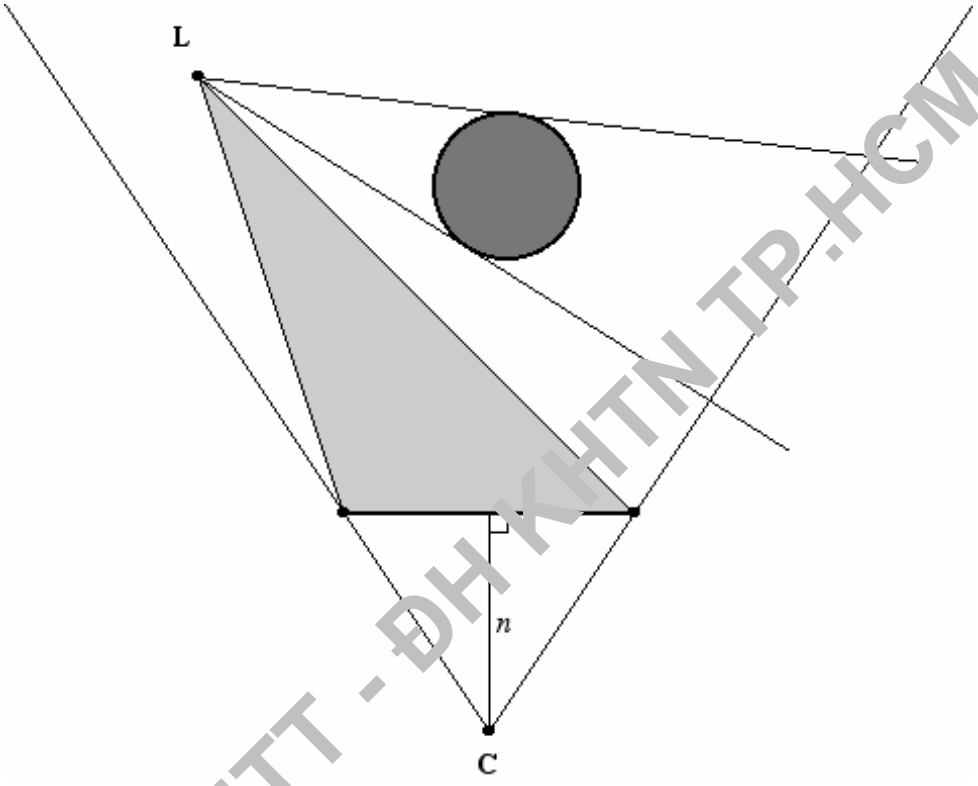
- Các vật chắn sáng với các mặt đã được chuẩn hóa về dạng lưới các tam giác, lưới này khép kín
- Thế giới cần dựng với các mặt đã được chuẩn hóa về dạng lưới các tam giác
- Bước 1 : Xóa frame buffer, render thế giới
- Bước 2 : Chọn nguồn sáng và các vật thể chắn sáng, nếu đây không phải nguồn sáng đầu tiên xóa stencil buffer
- Bước 3 : Đối với mỗi vật chắn sáng, tính toán silhouette của nó, render shadow volume bằng cách chiếu silhouette ra xa vô tận với tâm là nguồn sáng
- Bước 4 : Render shadow volume, sử dụng các toán tử stencil để đặt các giá trị stencil khác không trong stencil buffer nơi các bề mặt nằm trong vùng bóng
- Bước 5 : Chiếu sáng thế giới, sử dụng toán tử stencil đánh dấu các vùng nhìn thấy không được chiếu sáng
- Bước 6 : Lặp từ bước 2 tới bước 5 đối với mỗi nguồn sáng

Xác định khi nào cần sử dụng hai nắp đậy cho shadow volume

Như đã trình bày, khi shadow volume bị cắt bởi mặt phẳng near hay khi camera nằm trong shadow volume thì chúng ta phải render hai nắp đậy cho shadow volume.Do đó phải có một phương pháp tính toán để kiểm tra xem các trường hợp trên có xảy ra không

Gọi near regtangle là hình chữ nhật được cắt ra từ mặt phẳng near bởi 4 mặt của view frustum. Một cách để kiểm tra xem shadow volume có bị cắt bởi mặt phẳng near của view frustum hay không là xây dựng một tập các mặt phẳng nối từ các cạnh của near regtangle

tới nguồn sáng, gọi vùng không gian được bao bởi các mặt phẳng này và near regtangle là near-clip volume. Chỉ những điểm nằm trong near-clip volume khi bị đẩy ra xa vô tận mới cắt near regtangle, vì thế khi vật chắn sáng hoàn toàn nằm ngoài near-clip volume chúng ta không cần render hai nắp của shadow volume



Vật chắn sáng nằm ngoài near-clip volume, không cần render hai nắp của shadow volume

Khi xây render near-clip volume chúng ta cần phải xét 3 trường hợp

- Nguồn sáng nằm phía trước mặt phẳng near
- Nguồn sáng nằm phía sau mặt phẳng near
- Nguồn sáng nằm rất gần mặt phẳng near

Gọi **W** là ma trận chuyển đổi ánh xạ từ hệ tọa độ mắt nhìn sang hệ tọa độ thế giới thực, **L** là tọa độ của nguồn sáng (thành phần $\mathbf{L_w}$ = 1). Nguồn sáng được xem như nằm trong mặt phẳng near nếu khoảng cách từ nó tới mặt phẳng near nhỏ hơn một giá trị dương **d**

cho trước, khi nguồn sáng nằm ở xa vô tận ($\mathbf{L_w} = 0$) thì khoảng cách từ đó tới mặt phẳng near coi như chiều dài của phép chiếu của vector chỉ hướng của nguồn sáng đã được chuẩn hóa tới vector chỉ hướng của mặt phẳng near. Trong các trường hợp trên chúng ta có thể tính được khoảng cách **d** từ nguồn sáng tới mặt phẳng near cho bởi công thức :

$$d = \left(\mathbf{W}^{-1}\mathbf{L}\right) \cdot \langle 0, 0, -1, -n \rangle .$$

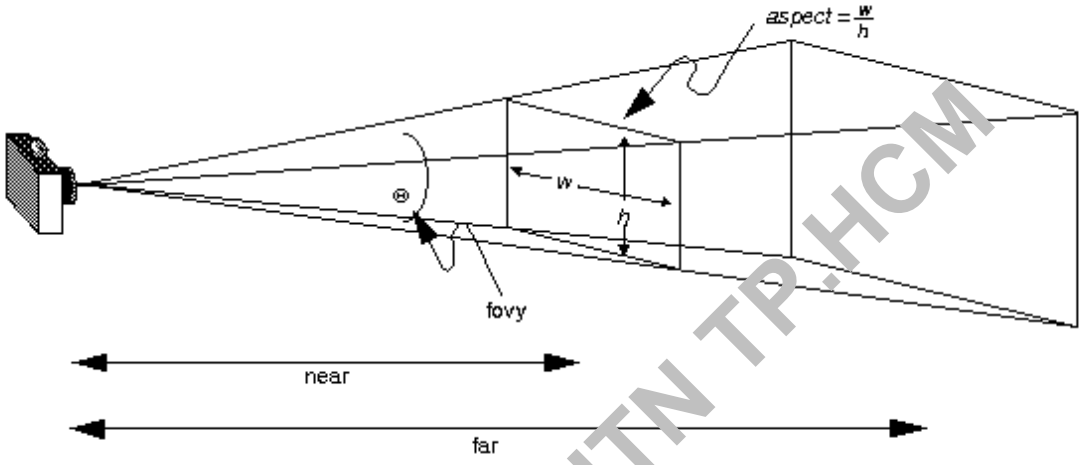
Nếu **d** > δ nguồn sáng nằm trước mặt phẳng near, **d** <- δ nguồn sáng nằm sau mặt phẳng near, trong trường hợp khác nguồn sáng xem như nằm trong mặt phẳng near

Trong trường hợp nguồn sáng nằm trong mặt phẳng near, near-clip volume được xác định bởi 2 mặt phẳng, $\mathbf{K_0}$ = <0,0,-1,-n> và $\mathbf{K_1}$ = <0,0,1,n>, hai mặt phẳng này trùng nhau nhưng chỉ khác về hướng của vector chỉ hướng. Từ đây ta có thể render được near-clip volume, và việc kiểm ta xem vật chắn sáng có nằm trong near-clip volume hay không chính là việc xem nó có bị cắt bởi mặt phẳng near hay không

Trong trường hợp nguồn sáng không nằm trong mặt phẳng near, ta cần tính toán các đỉnh của hình chữ nhật near rectangle. Trong hệ tọa độ mắt nhìn 4 đỉnh của near regtangle R_0, R_1, R_2, R_3 lần lượt là

$$\mathbf{R}_0 = \langle n/e, an/e, -n \rangle$$
$$\mathbf{R}_1 = \langle -n/e, an/e, -n \rangle$$
$$\mathbf{R}_2 = \langle -n/e, -an/e, -n \rangle$$
$$\mathbf{R}_3 = \langle n/e, -an/e, -n \rangle$$

trong đó n là khoảng cách từ camera tới mặt phẳng near, a là tỷ lệ của viewport (width/height), e là chiều dài tiêu cự của nguồn sáng, e = 1/tan(α/2) với α là góc chiếu (field of view – fov) của view frustum như hình vẽ



Đó là 4 đỉnh trên sắp thứ tự theo ngược chiều kim đồng hồ. Trong trường hợp nguồn sáng nằm phía trước mặt phẳng near, các vector chỉ hướng của bốn mặt near-clip volume trong hệ tọa độ thế giới thực N_i với 0 <= i <= 3 được cho bởi tích hữu hướng:

$$\mathbf{N}_i = \left(\mathbf{R}'_i - \mathbf{R}'_{(i-1) \bmod 4}\right) \times \left(\left\langle L_x, L_y, L_z \right\rangle - L_w \mathbf{R}'_i\right),$$

Với mỗi R'_i là một đỉnh của near rectangle trong hệ tọa độ thế giới thực, $R'_i = W*R_i$. Trong trường hợp nguồn sáng nằm sau mặt phẳng near, các vector chỉ hướng của hệ tọa độ thế giới thực được đổi dấu. Các mặt phẳng bao quanh near-clip volume trong hệ tọa độ thế giới thực K_i được tính như sau :

$$\mathbf{K}_i = \frac{1}{\|\mathbf{N}_i\|} \left\langle (\mathbf{N}_i)_x, (\mathbf{N}_i)_y, (\mathbf{N}_i)_z, -\mathbf{N}_i \cdot \mathbf{R}'_i \right\rangle.$$

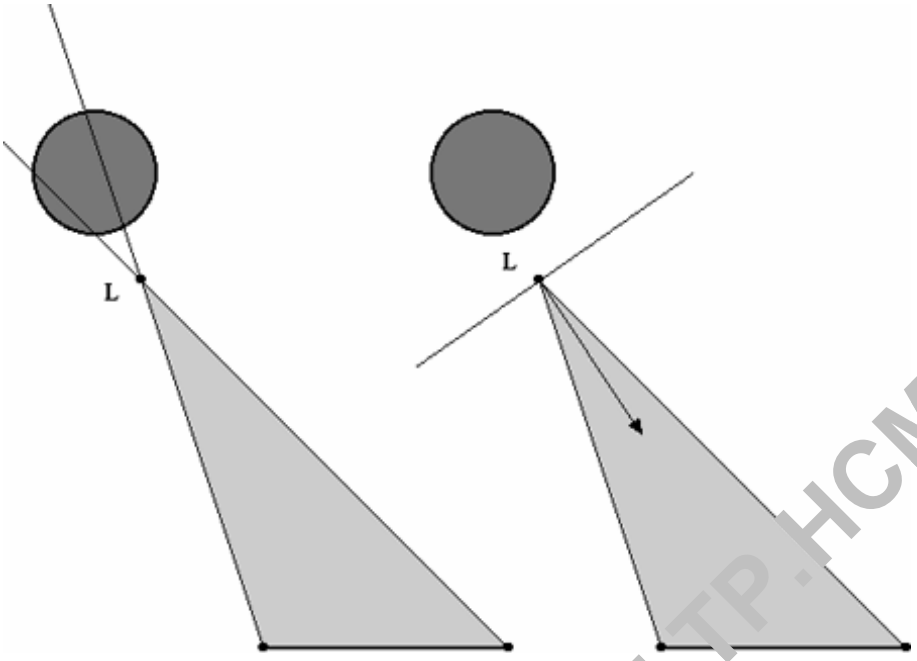
Thêm một mặt phẳng thứ 5 cho near-clip volume để nó trở thành một không gian khép kín, mặt phẳng này trùng với mặt phẳng near và có vector chỉ hướng hướng về nguồn sáng. Trong trường hợp nguồn sáng nằm phía trước mặt phẳng near mặt phẳng thứ 5 này (K_4) cho bởi

$$\mathbf{K}_4 = \left(\mathbf{W}^{-1}\right)^{\mathrm{T}} \langle 0, 0, -1, -n \rangle,$$

Trong trường hợp nguồn sáng nằm phía sau mặt phẳng near mặt phẳng thứ 5 này (K_4) được phủ định từ công thức trên

Chúng ta kiểm tra xem vật chắn bóng có hoàn toàn nằm trong near-clip volume hay không bằng cách xem hình bao của vật thể với các mặt phẳng K_i. Nếu hình bao của vật thể hoàn toàn nằm phía ngoài của bất kỳ mặt phẳng K_i nào shadow volume tạo ra từ vật thể và nguồn sáng không bị cắt bởi mặt phẳng near. Trong trường hợp chúng ta chọn hình bao của vật chắn sáng là một hình cầu tâm C, bán kính r, không cần render hai nắp của shadow volume nếu K_i *C < -r với mọi i

Trong trường hợp vật chắn sáng nằm phía sau nguồn sáng, lỗi có thể xảy ra khi hình bao nằm ngoài near-clip volume nhưng không hoàn toàn nằm ngoài bất kỳ mặt phẳng K_i nào, lúc này một phần của vật chắn sáng sẽ bị đổ bóng.



Lỗi xảy ra khi hình bao nằm ngoài near-clip volume nhưng không hoàn toàn nằm ngoài bất kỳ mặt phẳng K_i nào, ta thêm vào một mặt phẳng để phân loại

Chúng ta có thể giải quyết trường hợp này bằng cách thêm một mặt phẳng cho near-clip volume tại nguồn sáng và có vector chỉ hướng hướng về trung tâm của near regtangle như hình vẽ trên. Vector chỉ hướng N_5 được tính bởi :

$$\mathbf{N}_5 = \left(\mathbf{W}^{-1}\right)^{\mathrm{T}} \langle 0, 0, -n, 1 \rangle - \mathbf{L},$$

và mặt phẳng tương ứng K5 là

$$\mathbf{K}_5 = \frac{1}{\|\mathbf{N}_5\|} \left\langle (\mathbf{N}_5)_x, (\mathbf{N}_5)_y, (\mathbf{N}_5)_z, -\mathbf{N}_5 \cdot \mathbf{L} \right\rangle.$$

4. Ưu điểm

Shadow volume có thể đổ bóng cho các vật thể lên các bề mặt bất kỳ của vật hứng bóng, thuật toán này rất thích hợp cho việc đổ bóng cho toàn bộ các đối tượng trong một thế giới thực phức tạp

Các vật thể hứng bóng hay chắn sáng, cho dù phức tạp, đều có thể tự đổ bóng nội tại trong bản thân mình

Không phụ thuộc vào số lượng đa giác của vật hứng bóng

Không phụ thuộc vào tính chất hình học của bóng

5. Khuyết điểm

Chi phí tính toán phức tạp, nhất là chi phí tính toán silhouette

Đòi hỏi cấu hình phần cứng : phải hỗ trợ stencil buffer

6. Nhận xét

Đây là thuật toán tạo bóng đang được ưa chuộng hiện nay, nó có thể được áp dụng trong mọi địa hình, mọi thế giới có các vật thể hình học phức tạp. Thuật toán có thể tự đổ bóng được trên vật chắn sáng và đổ bóng được tại mọi nơi trong thế giới nên cho ra bóng có độ chính xác cao và trông thực tế hơn. Tuy nhiên kỹ thuật này đòi hỏi phải được sự hỗ trợ từ phần cứng

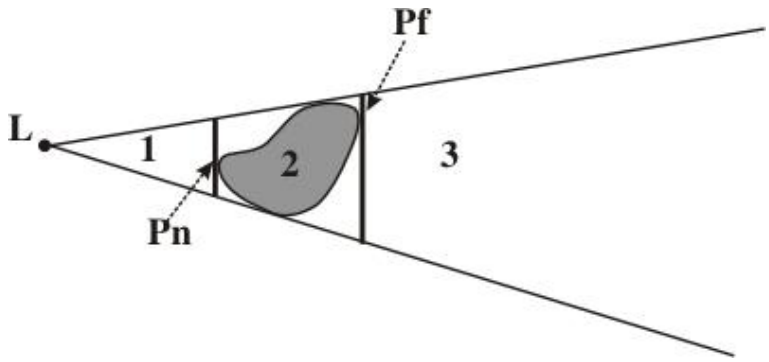
III. PROJECTIVE SHADOW MAPPING

1. Ý tưởng chính

Trong thuật toán shdow volume việc tính toán silhouette tốn kém chi phí tương đối cao và điều quan trọng là không thể hiện được của các vật hứng sáng có một độ trong suốt nhất định như kính, màn mỏng. Một cách tiếp cận mới là tìm ra hình chiếu của vật chắn sáng trong phép chiếu từ nguồn sáng đưa hình chiếu này thành texture và ánh xạ hình chiếu này lên các vật hứng bóng

Một vật thể chắn một nguồn sáng cho ra một vùng không gian shadow volume, ta sẽ khảo sát xem hình dạng phần cắt ngang của shadow volume sẽ thay đổi như thế nào khi thay đổi khoảng cách của vật thể và nguồn sáng. Trước tiên, chia shadow volume thành 3 phần như sau:

- Phần giữa nguồn sáng và điểm Pn (điểm gần nguồn sáng nhất của vật thể).
- Phần giữa Pn và Pf (điểm xa nguồn sáng nhất của vật thể)
- Phần từ Pf đến vô cùng.

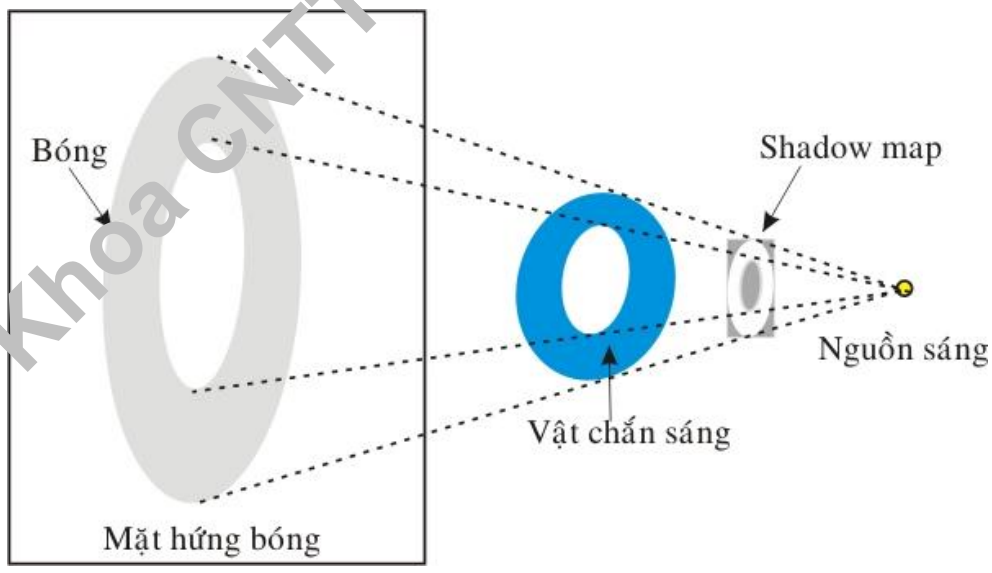


Ta dễ dàng thấy rằng phần cắt ngang shadow volume trong phần 3 luôn luôn có dạng cố định, chỉ có kích thước là thay đổi (càng xa nguồn sáng, kích thước càng tăng). Nhờ tính chất này, trừ phi vật thể hứng bóng nằm trong vùng 2, shadow volume có thể được render chính xác bằng cách chiếu một mặt nạ hai chiều từ vị trí nguồn sáng, Vì vậy, ta có thể ánh xạ mặt nạ 2 chiều này lên các vật nhận khác nhau để xác định những vùng có bóng mà chỉ cần dùng cùng một phép chiếu. Mặt nạ 2 chiều này được gọi là shadow map. Shadow map được tạo ra bằng cách vẽ silhouette của vật thể được nhìn từ nguồn sáng. Phương pháp này được gọi là projective shadow mapping

2. Tạo shadow map

Shadow map

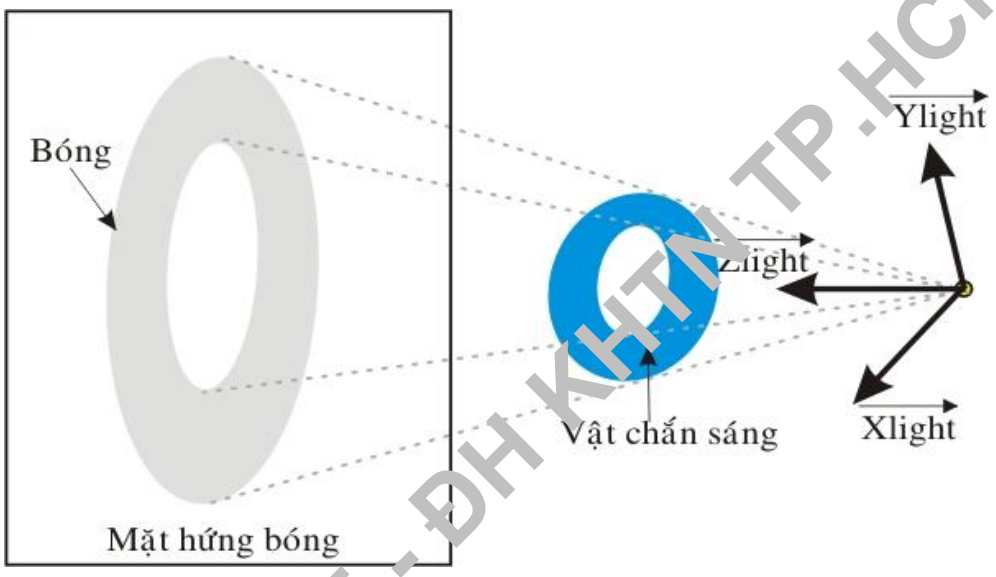
Đầu tiên ta thiết lập một phép chiếu phối cảnh với tâm là nguồn sáng. Phép chiếu này chiếu vật chắn sáng lên trên một mặt phẳng ảo ở giữa nguồn sáng và vật thể tạo ra shadow map của vật thể đó



Chiếu shadow map

Hệ tọa độ nguồn sáng

Để thực hiện phép chiếu trên, trước tiên ta định nghĩa một hệ tọa độ có gốc là nguồn sáng và trục Z hướng về phía vật chắn sáng. Trục Z của hệ tọa độ này sẽ xác định đường tâm của phép chiếu trong khi đó mặt phẳng tạo bởi 2 trục X-Y xác định các trục của mặt phẳng ảo mà chúng ta chiếu shadow map lên đó. Nếu chuyển vật chắn sáng sang hệ tọa độ nguồn sáng ta có thể chiếu nó lên mặt phẳng ảo một cách dễ dàng



Hệ tọa độ nguồn sáng

Để render một hệ tọa độ bất kỳ, chúng ta cần phải biết được gốc và các trục của nó. Trong hệ tọa độ nguồn sáng, chúng ta đã biết được gốc là nguồn sáng , các trục còn lại được gọi lần lượt là X_{light}, Y_{light}, Z_{light}, tất cả các vector còn lại đều thuộc về hệ tọa độ thế giới thực.

Xác định Z_{light}

Đầu tiên chúng ta xác định trục Z_{light}, các trục X_{light} và Y_{light} có thể được xác định dễ dàng từ Z_{light}. Z_{light} là một vector có hướng có gốc là nguồn sáng và hướng về phía vật chắn sáng, ta có thêm tập đỉnh của vật chắn sáng từ đó trục Z_{light} được xác định nhanh chóng bằng cách

tính trung bình các vector có hướng có gốc là nguồn sáng và điểm còn lại là một đỉnh của vật chắn sáng. Vector có hướng trung bình (ký hiệu là MDV – mean direction vector) được tính như sau

$$\overrightarrow{MDV} = \frac{\sum_{i=1}^{N_v}(V_i - P_{light})}{N_v}$$

Trong đó N_v là số đỉnh của vật thể và P_{light} là vị trí của nguồn sáng. Chuẩn hóa vector MDV ta được Z_{light}

$$\vec{z}_{light} = \frac{\overrightarrow{MDV}}{|\overrightarrow{MDV}|}$$

Xác định X_{light} và Y_{light}

Trong phép chiếu phối cảnh để tạo ra shadow map, X-Y_{light} không ảnh hưởng tới phép chiếu, nghĩa là nếu ta xoay X-Y_{light} quanh Z_{light} phép chiếu vẫn không thay đổi. Từ đó ta có thể chọn trục X_{light} là một vector đơn vị bất kỳ vuông góc với trục Z_{light}, trục Y_{light} sẽ tìm được bằng tích hữu hướng của X_{light} và Z_{light}. X_{light} có thể tìm được bằng cách nhân hữu hướng Z_{light} với một vector V không song song với nó. Để cho đơn giản, ta cho vector V(x,y,z) là vector đơn vị trong hệ tọa độ thế giới thực với một tọa độ bằng 1, hai tọa độ còn lại bằng 0.

Trong 3 tọa độ (X, Y, Z) của một vector, tọa độ nào lớn nhất sẽ chi phối nhiều nhất đến hướng của vector đó. Do đó, để có một vector trỏ ra xa vector Z_{light} nên ta sẽ tìm thành phần trong V tương ứng với thành phần có giá trị tuyệt đối nhỏ nhất trong Z_{light} và đặt giá trị cho nó là 1

Ví dụ

- Nếu Z_{light} = (0.381, 0.889, 0.254) thì V = (0.0, 0.0, 1.0)
- Nếu Z_{light} = (-0.889, 0.254, 0.381) thì V = (0.0, 1.0, 0.0)

Kết quả của tích hữu hướng giữa Z_{light} và V là một vector vuông góc với cả hai, sau khi chuẩn hóa ta được X_{light}

$$\overrightarrow{X}_{light} = \frac{\overrightarrow{Z}_{light} \times \overrightarrow{V}}{\left| \overrightarrow{Z}_{light} \times \overrightarrow{V} \right|}$$

Sau khi đã có X_{light} và Z_{light} ta tính Y_{light}

$$\overrightarrow{Y}_{light} = \overrightarrow{X}_{light} \times \overrightarrow{Z}_{light}$$

Vì X-Z_{light} là các vector đơn vị nên chúng ta không cần chuẩn hóa Y_{light}
Từ Xlight, Ylight, Zlight, Plight đã biết, ta có thể tạo ra được ma trận ánh xạ một điểm từ hệ tọa độ thế giới thực sang hệ tọa độ nguồn sáng

$$M_{WorldToLight} = \begin{bmatrix} X_{of}\overrightarrow{X}_{light} & X_{of}\overrightarrow{Y}_{light} & X_{of}\overrightarrow{Z}_{light} & 0.0 \\ Y_{of}\overrightarrow{X}_{light} & Y_{of}\overrightarrow{Y}_{light} & Y_{of}\overrightarrow{Z}_{light} & 0.0 \\ Z_{of}\overrightarrow{X}_{light} & Z_{of}\overrightarrow{Y}_{light} & Z_{of}\overrightarrow{Z}_{light} & 0.0 \\ -X_{of}\overrightarrow{P}_{light} & -Y_{of}\overrightarrow{P}_{light} & -Z_{of}\overrightarrow{P}_{light} & 1.0 \end{bmatrix}$$

Bước kế tiếp là tính ma trận để chuyển đổi từ hệ tọa độ của vật chắn sáng sang hệ trục tọa độ nguồn sáng. Ta có :

$$M_{BlockerLocalToLight} = M_{BlockerLocalToWorld} * M_{WorldToLight}$$

Với $M_{BlockerLocalToLight}$ là ma trận để chuyển đổi từ hệ tọa độ của vật chắn sáng sang hệ trục tọa độ nguồn sáng, $M_{BlockerLocalToWorld}$ là ma trận để chuyển đổi từ hệ tọa độ của vật chắn sáng sang hệ trục tọa độ

thế giới thực, $M_{WorldToLight}$ là ma trận để chuyển đổi từ hệ tọa độ thế giới thực sang hệ trục tọa độ nguồn sáng. Vì mặt phẳng ảo dùng trong phép chiếu tạo shadow map song song với mặt phẳng $X\text{-}Y_{light}$ nên ta cũng có thể xác định được một ma trận chiếu để tạo ra shadow map

Xác định phép chiếu phối cảnh

Để xác định phép chiếu ta cần góc chiếu hay tỷ lệ chiếu trên các trục X, Y. Các tỷ lệ chiếu R_X, R_Y cho mỗi đỉnh của vật chắn sáng được xác định qua phép biến đổi với ma trận $M_{BlockerLocalToLight}$ và chia các kết quả trên trục X, Y cho trục Z

$$R_Y = \left| \frac{V_{tx_Y}}{V_{tx_Z}} \right|$$

Phép chiếu thích hợp

Chúng ta luôn luôn thực hiện phép chiếu này để tạo shadow map, tuy nhiên silhouette của vật chắn sáng bị thay đổi khi ta thay đổi vị trí của nguồn sáng ra xa hoặc lại gần nó hay khi kích cỡ của vật chắn sáng bị thay đổi. Sự thay đổi của silhouette có thể dẫn tới một ảnh chiếu rất nhỏ của silhouette ở chính giữa shadow map hay lớn hơn shadow map, kích thước của silhouette của vật chắn sáng sẽ không vừa khớp với shadow map, khi chiếu lên vật hứng bóng sẽ cho ra bóng không thích hợp

A **B** **C**

Không thích hợpA-B, thích hợp C

Để thực hiện phép chiếu thích hợp sao cho ánh xạ được vật chắn sáng vừa vặn vào shadow map ta sử dụng giá trị lớn nhất của R_X (R_{Xmax}) làm tỉ lệ ngang của phép chiếu và R_Y (R_{Ymax}) làm tỉ lệ dọc của phép chiếu. Ta tìm được ma trận chiếu như sau

$$M_{Blocker Projection} = \begin{bmatrix} \frac{0.98}{R_{X\max}} * SMapWidth * \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{0.98}{R_{Y\max}} * SMapHeight * \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Với SMapWidth và SMapHeight là độ phân giải theo chiều ngang và dọc của shadow map, Z_{near}, Z_{far} là khoảng cách tới mặt phẳng near và far của view frustum của nguồn sáng.

Nhân ma trận này với ma trận $M_{BolckerLocalToLight}$ ta được ma trận của phép chiếu một vật thể từ hệ tọa độ cục bộ thành hệ tọa độ texture của Shadow Map

$$M_{BolckerLocalToShadowMap} = M_{BolckerLocalToLight} * M_{BolckerProjection}$$

3. Chiếu Shadow map lên vật hứng bóng

Từ các bước xử lý bên trên ta đã có shadow map của vật chắn sáng ứng với một nguồn sáng. Shadow map này có thể chiếu lên trên một hay

nhiều vật hứng bóng bất kỳ cho dù vật hứng bóng đó có hình dáng như thế nào bởi vì shadow map được áp lên như một texture

Chúng ta sẽ sử dụng lại phép chiếu để tạo shadow map để chiếu shadow map lên các vật hứng bóng. Ma trận chiếu được thay đổi 2 hệ số viền và tỷ lệ để phù hợp cho việc chiếu shadow map lên vật hứng bóng, ta có ma trận chiếu shadow map lên vật hứng bóng $M_{ReceiverProjection}$ như sau

$$M_{ReceiverProjection} = \begin{bmatrix} \frac{0.49}{R_{X\max}} * SMapWidth * \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{0.49}{R_{Y\max}} * SMapHeight * \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ -0.5 & -0.5 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Nhân ma trận này với ma trận $M_{WorldToLight}$ ta được ma trận của phép chiếu từ hệ tọa độ thế giới thực sang hệ tọa độ texture của Shadow Map

$$M_{WorldToShadowMap} = M_{WorldToLight} * M_{ReceiverProjection}$$

Nhân tiếp ma trận $M_{WorldToShadowMap}$ này với các ma trận $M_{ReceiverLocalToWorld}$ để được ma trận của phép chiếu từ hệ tọa độ cục bộ của các vật hứng sáng sang hệ tọa độ texture của Shadow Map

$$M_{ReceiverLocalToShadowMap} = M_{ReceiverLocalToWorld} * M_{WorldToShadowMap}$$

Chúng ta sử dụng các hệ số viền là 0.98 và 0.49 thay cho 1 và 0.5 trong hai ma trận $M_{BlockerProjection}$ và $M_{ReceiverProjection}$ để không ánh xạ shadow map lên viền của texture bóng trên vật hứng bóng, tránh được hiện tượng tự động lặp lại texture trên viền trên toàn bộ bề mặt vật hứng bóng.

Render vật hứng bóng

- Cách render Single-pass : Nếu vật hứng bóng không có texture chúng ta render vật thể với texture là shadow map đen trắng và chiếu sáng vật hứng bóng
- Cách render Multi-pass : Nếu vật hứng bóng bản thân đã có sẵn texture chúng ta phải dán thêm texture shadow map lên vị trí thích hợp của nó. Render các đối tượng nhiều lần với các texture khác nhau rồi trộn chúng lại theo các cách thức khác nhau. Nếu phần cứng có hỗ trợ kỹ thuật multitexture chúng ta có thể dán texture shadow map vào, sau đó blend hai texture này với nhau.

4. Ưu điểm

Đổ bóng được trên các mặt cong, các mặt hình học phức tạp

Do quá trình tạo shadow map là quá trình render bình thường nên có thể render vật chắn sáng với độ trong suốt hay độ mờ đục

5. Khuyết điểm

Tốn thêm thời gian cho quá trình render shadow map:

- Thời gian xoá frame buffer
- Thời gian sao chép dữ liệu từ frame buffer sang bộ nhớ chính (cả CPU và GPU phải đồng thời xử lý).

Chất lượng bóng phụ thuộc vào

- Kích thước texture làm shadow map, quá trình render vật chắn sáng tạo shadow map : Có thể xảy ra hiện tượng hình ảnh vật chắn sáng chỉ nằm một phần nhỏ trong shadow map dẫn đến lãng phí bộ nhớ nhưng chỉ đạt được chất lượng bóng tương đương với shadow map có độ phân giải thấp hoặc các đường cạnh của vật

chắn sáng không song song với phương ngang và phương thẳng đứng => hình thành các đương biên nghiêng gây ra răng cưa trong shadow map

- Độ xa gần của bề mặt đổ bóng
- Kích thước của vật chắn sáng

Thời gian render bóng phụ thuộc vào vật hứng bóng. Vì bản chất của chức năng multitexture là quá trình render các đối tượng nhiều lần với các texture khác nhau rồi trộn chúng lại theo các cách thức khác nhau nên quá trình render bóng lên các đối tượng hình học phức tạp lâu hơn trên các bề mặt đơn giản

Không có khả năng tự đổ bóng lên chính vật thể chắn sáng

Có hiện tượng đổ bóng ngược

6. Nhận xét

Đâ là thuật toán thường được áp dụng trong các thế giới có các vật dụng thủy tinh và các bề mặt đơn giản, vì mặc dù có thể đổ bóng được trên các vật thể phức tạp tuy nhưng tốn kém chi phí, hơn nữa bóng phụ thuộc vào kích thước shadow map nên nếu muốn bóng chất lượng cao phải tốn bộ nhớ để tăng kích thước texture.

PHẦN 4 : KẾT QUẢ CÀI ĐẶT VÀ ỨNG DỤNG

Luận văn đã xây dựng được một thế giới ba chiều tương đối hoàn hảo và giống với thực tế, cho phép người dùng có thể tương tác được với thế giới. Các hiệu ứng về mặt ánh sáng, tính chất vật lý rơi, dụng chạm và hiệu ứng bóng đã được cài đặt để làm tăng tính trung thực và tăng sự cảm nhận chính xác của người dùng đối với thế giới

I. HỆ THỐNG ĐIỀU KHIỂN

Chuột : Quay góc nhìn

W : Di chuyển tới

S : Di chuyển lui

A : Di chuyển sang trái

D : Di chuyển sang phải

H / F1 : Bật màn hình trợ giúp

P : Chuyển đổi qua lại giữa các trạng thái không bóng, shadow volume, planar shadow và projective shadow mapping

C : Bật tắt chế độ xét va chạm giữa camera với thế giớI

> / < : Tăng giảm tốc độ

V : Bật tắt chế độ quan sát silhouette/ shadow volume/ shadow đối với thuật toán shadow volume

I : Thay đổi kích thước shadow map

L : Bật tắt dao động của đèn

II. YÊU CẦU

CPU Pentium III 750 MHz

Hệ điều hành : Windows ME, Windows 2000, Windows XP

Bộ nhớ 128 MB RAM

Card đồ họa GeForce3 32MB bộ nhớ(do đề tài có sử dụng một hàm mở rộng của phần cứng là GenOcclusionQueriesNV chỉ nằm trong phần cứng card đồ họa GeForce 3 trở lên)

Driver card đồ họa có hỗ trợ OpenGL

III. ĐÁNH GIÁ VÀ KẾT LUẬN

Luận văn đã hoàn thành mục tiêu ban đầu được đề ra là nghiên cứu và cài đặt hoàn chỉnh 3 thuật toán tạo bóng phổ biến và hiệu quả hiện nay là planar shadow, shadow volume và projective shadow mapping. Không chỉ dừng lại ở các chương trình minh họa đơn giản cục bộ, có thể cài đặt chỉ bằng các thuật toán tạo bóng nguyên thủy, đề tài đã tập trung nghiên cứu các ưu khuyết điểm, các hướng cải tiến mới nhất của các thuật toán tạo bóng để các thuật toán tạo bóng này có thể được áp dụng vào một thế giới

ba chiều tương tác rộng lớn thời gian thực. Điều này được chứng minh qua việc cài đặt thành công 3 thuật toán tạo bóng trong mô hình nhà tù Chuồng Cọp thuộc Bảo Tàng Di Tích Chiến Tranh tại thành phố Hồ Chí Minh.

Trong các trò chơi ba chiều phổ biến hiện nay, việc tạo bóng cho nhân vật và các vật thể trong thế giới vẫn là một bài toán chưa được giải quyết trọn vẹn do chưa khắc phục triệt để được các khuyết điểm của các thuật toán tạo bóng khi áp dụng vào trong môi trường ứng dụng thực tế. Các trò chơi ba chiều được các công ty nổi tiếng phát triển hiện nay như : NeverWinter Nights năm 2002, Tiger Wood 2002, Enter the Matrix năm 2003, … vẫn còn bộc lộ một số hạn chế và sai sót trong kỹ thuật tạo bóng. Luận văn này đã đưa ra được một hướng giải quyết tương đối phù hợp cho việc đưa các thuật toán tạo bóng vào các chương trình đồ họa ba chiều tương tác thời gian thực. Trong hướng giải quyết này chúng tôi đã kết hợp thuật toán nguyên thủy với các cải tiến xuất phát từ nhiều công trình nghiên cứu khoa học trên thế giới

Tuy thời gian thực hiện còn hạn hẹp, luận văn này đã đạt được một thành công đáng khích lệ nhất định. Có thể nói chúng tôi đã thành công khi bắt tay vào một đề tài còn mới mẻ ở Việt Nam nói riêng và vẫn còn đang là một trong những đề tài được quan tâm, nghiên cứu nhiều nhất của công nghệ đồ họa ba chiều trên thế giới nói chung. Thông qua quá trình nghiên cứu chúng tôi nhận thấy rằng rằng nếu dành cho đồ họa ba chiều một sự quan tâm đúng mức, chúng ta, những người Việt Nam, hoàn toàn đủ sức để khai phá lĩnh vực này, thậm chí cả với những công nghệ tiên tiến nhất

Mô hình hiện nay của luận văn có thể được mở rộng theo nhiều hướng để tạo ra các chương trình ứng dụng thực tế như : Tham quan Bảo Tàng Di Tích Chiến Tranh, các mô hình thế giới trong các trò chơi, các mô hình mô phỏng,… đây chính là mục tiêu phấn đấu của chúng tôi trong tương lai

PHỤ LỤC

GIÁ TRỊ HIỆN HÀNH VÀ DỮ LIỆU LIÊN QUAN

Biến giá trị	Khởi tạo mặc định	Hàm lấy giá trị
GL_STENCIL_TEST	GL_FALSE	glIsEnable()
GL_STENCIL_FUNC	GL_ALWAYS	glGetIntegerv()
GL_STENCIL_VALUE_MASK	1's	glGetIntegerv()
GL_STENCIL_REF	0	glGetIntegerv()
GL_STENCIL_FAIL	GL_KEEP	glGetIntegerv()
GL_STENCIL_PASS_DEPTH_FAIL	GL_KEEP	glGetIntegerv()
GL_STENCIL_PASS_DEPTH_PASS	GL_KEEP	glGetIntegerv()
GL_STENCIL_WRITEMASK	1's	glGetIntegerv()

GL_STENCIL_CLEAR_VALUE	0	glGetIntegerv()
GL_DEPTH_TEST	GL_FALSE	glIsEnable()
GL_DEPTH_FUNC	GL_LESS	glGetIntegerv()

CÁC MA TRẬN CHUYỂN ĐỔI

Tịnh tiến :

Tịnh tiến tới tọa độ (x,y,z,1)

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ma trận khả nghịch :

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Biến đổi tỷ lệ :

Thay đổi tỷ lệ x,y,z trên các trục

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ma trận khả nghịch :

$$S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quay :

Quay quanh trục x một góc *a* :

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quay quanh trục y một góc *a* :

$$R_y = \begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quay quanh trục z một góc *a* :

$$R_z = \begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Phép chiếu phối cảnh :

Phép chiếu phối cảnh với chóp nhìn (view frustum) có các giá trị trái l, phải r, dưới b, trên t, gần n, xa f :

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Ma trận khả nghịch :

$$P^{-1} = \begin{bmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{bmatrix}$$

Phép chiếu song song :

Phép chiếu song song với khối nhìn (view volume) có các giá trị trái l, phải r, dưới b, trên t, gần n, xa f :

$$o = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ma trận khả nghịch :

$$o^{-1} = \begin{bmatrix} \frac{r-l}{2} & 0 & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t-b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TÀI LIỆU THAM KHẢO

[1] : Ashu Rege, Occlusion (HP and NV Extensions) (HP and NV Extensions), NVIDIA Corporation, 1/2003

[2] : Tom Hall, Silhouette Tracking, www.geocities.com/tom_j_hall, 5/2003

[3] : Eric Lengyel, The Mechanics of Robust Stencil Shadows, Gamasutra, http://www.gamasutra.com/features/20021011/lengyel_01.htm, 10/2002

[4] : Yen Kwoon Hun, The Theory of Stencil Shadow Volumes, Gamedev, http://www.gamedev.net/reference/articles/article1873.asp , 3/2002

[5] **:** Tom Davis, Dave Shreiner, Mason Woo và Jackie Neider, OpenGL Programming Guide, Third Edition, Addison Wesley Longman Inc, 1999

[6] : Tomas Moller và Eric Haines, Realtime Rendering, A K Peters Ltd, Natick, 1999

[7] : Mark J. Kilgard, Improving Shadows and Reflections via the Stencil Buffer, NVIDIA Corporation, 7/2000

[8] : Mark A.DeLoura, Game Programming Gem, Charles River Media Inc, Rockland, Massachusetts, 2000

[9] : Mark J.Kilgard, Shadow Mapping with Today's OpenGL Harware, NVIDIA Corporation, 2001

[10] : Cass Everitt và Mark J. Kilgard, Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering, NVIDIA Corporation, 12/2002

[11] : Tiago Sousa, Real Time Shadow Techniques, Real Time Computer Graphics, 1/2003

[12] : D. Sim Dietrich Jr, Projective Shadows Shadows, NVIDIA Corporation, 12/2002

[13] : Tiago Sousa, Projective Texturing based FX, Real Time Computer Graphics, 12/2002

[14] : Các website nói về vấn đề tạo bóng trong đồ họa ba chiều : www.nehe.gamedev.com, www.gameprogramming.com, www.gamedev.net , www.gamasutra.com, www.opengl.org.