BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

-------o0o-------

ISO 9001:2008

TÌM HIỂU KỸ THUẬT TẠO BÓNG CỨNG SHADOW VOLUME

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ Thông tin

Sinh viên thực hiện: Nguyễn Tiến Dũng

Giáo viên hướng dẫn: PGS.TS. Đỗ Năng Toàn

Mã số sinh viên: 1351010036

HẢI PHÒNG – 2013

LỜI CẢM ƠN

Em xin gửi lời cảm ơn tới các thầy cô khoa Công nghệ thông tin trường Đại học Dân Lập Hải Phòng, 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.

Em xin gửi lời cảm ơn sâu sắc tới thầy Đỗ Năng Toàn, người đã tận tình chỉ bảo và hướng dẫn chúng em thực hiện tốt đồ án 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 đồ án tốt nghiệp này.

Mục Lục

LỜI NÓI ĐẦU 3

CHƯƠNG 1: KHÁI QUÁT VỀ ĐỒ HỌA BA CHIỀU VÀ BÀI TOÁN TẠO BÓNG .. 4

1.1. Khái quát về đồ họa 3 chiều 4

1.1.1. Hiển thị 3D(3D Viewing) 4

1.1.2 Bộ đệm và các phép kiểm tra 11

1.2 Bài toán tạo bóng 12

1.2.1 Bóng và các dạng nguồn sáng 12

1.2.2 Một số cách tiếp cận trong tạo bóng 17

CHƯƠNG 2: KỸ THUẬT TẠO BÓNG CỨNG BẰNG PHƯƠNG PHÁP SHADOW VOLUME 19

2.1 Giới thiệu 19

2.2 Tìm danh sách cạnh viền. 20

2.3 Xác định các tứ giác bao quanh bóng khối 24

2.4 Tạo bóng khối bằng thuật toán Z-Pass. 27

2.5 Tạo bóng bằng thuật toán Z-Fail 30

2.5.1. Tất cả các mặt trước của bóng từ vị trí điểm nhìn 31

2.5.2 Tất cả các mặt sau của bóng từ vị trí điểm nhìn 32

2.5.3. Vẽ bóng đậy nắp 2 đầu của khối 33

2.6 So sánh giữa 2 thuật toán 35

CHƯƠNG III: CHƯƠNG TRÌNH THỰC NGHIỆM 36

3.1 Bài toán 36

3.2 Phân tích, lựa chọn công cụ 36

3.2.1 Giới thiệu ngôn ngữ lập trình 36

3.2.2 Lựa chọn công cụ 37

3.3 Kết quả chương trình 37

KẾT LUẬN 39

TÀI LIỆU THAM KHẢO: 39

LỜI NÓI ĐẦU

Đồ họa máy tính là một lĩnh vực phát triển nhanh nhất trong tin học. Nó được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau thuộc về khoa học, kỹ nghệ, y khoa, kiến trúc và giải trí…

Năm 1966, Sutherland ở Học viện Công nghệ Massachusetts là người đầu tiên đặt nền bóng cho đồ họa 3D bằng việc phát minh ra thiết bị hiển thị trùm đầu (head-amounted display) được điều khiển bởi máy tính đầu tiên. Nó cho phép người nhìn có thể thấy được hình ảnh dưới dạng lập thể 3D. Từ đó đến nay đồ họa 3D trở thành một trong những lĩnh vực phát triển rực rỡ nhất của đồ họa máy tính.

Nó được ứng dụng rộng rãi trong hầu hết tất cả các lĩnh vực như Điện ảnh, Hoạt hình, kiến trúc và các ứng dụng xây dựng các mô hình thực tại ảo…..Và không thể không nhắc đến vai trò tối quan trọng của đồ họa 3D trong việc tạo ra các game sử dụng đồ họa hiện nay như Doom, Halflife….Có thể nói đồ họa 3D đã đang và sẽ tạo nên một nền công nghiệp game phát triển mạnh mẽ.

Mục đích chính của đồ họa 3D là tạo ra và mô tả các đối tượng, các mô hình trong thế giới thật bằng máy tính sao cho càng giống với thật càng tốt. Việc nghiên cứu các phương pháp các kỹ thuật khác nhau của đồ họa 3D cũng chỉ hướng đến một mục tiêu duy nhất đó là làm sao cho các nhân vật, các đối tượng, các mô hình được tạo ra trong máy tính giống thật nhất. Và một trong các phương pháp đó chính là tạo bóng cho đối tượng.

Xuất phát từ vấn đề này đồ án của em xây dựng gồm 3 chương:

CHƯƠNG 1: KHÁI QUÁT VỀ ĐỒ HỌA BA CHIỀU VÀ BÀI TOÁN TẠO BÓNG

Chương này nói về các kiến thức cơ bản về ánh sáng, về hiển thị 3D, về biểu diễn điểm và các phép biến đổi và khái quát các kỹ thuật tạo bóng.

CHƯƠNG 2: KỸ THUẬT TẠO BÓNG CỨNG BẰNG PHƯƠNG PHÁP SHADOW VOLUME

Chương này đi vào chi tiết về kỹ thuật tạo bóng cứng Shadow Volume và các dạng nguồn sáng.

CHƯƠNG 3: CHƯƠNG TRÌNH THỰC NGHIỆM

CHƯƠNG 1: KHÁI QUÁT VỀ ĐỒ HỌA BA CHIỀU VÀ BÀI TOÁN TẠO BÓNG

1.1. Khái quát về đồ họa 3 chiều

1.1.1. Hiển thị 3D(3D Viewing)

1.1.1.1. Tổng quan

Các đối tượng trong thế giới thực phần lớn là các đối tượng 3 chiều còn thiết bị hiển thị chỉ 2 chiều. Do vậy, muốn có hình ảnh 3 chiều ta cần phải giả lập.

Chiến lược cơ bản là chuyển đổi từng bước. Hình ảnh sẽ được hình thành từ từ, ngày càng chi tiết hơn.

Các đối tượng trong mô hình 3D được xác định với tọa độ thế giới. Cùng với các tọa độ của đối tượng, người dùng cũng phải xác định vị trí và hướng của camera ảo trong không gian 3D và xác định vùng nhìn (là một vùng không gian được hiển thị trên màn hình)

Việc chuyển từ các tọa độ thế giới sang tọa độ màn hình được thực hiện theo 3 bước :

Bước đầu tiên thực hiện một phép biến đổi để đưa camera ảo trở về vị trí và hướng tiêu chuẩn. Khi đó điểm nhìn (eyepoint) sẽ được đặt ở gốc tọa độ, hướng nhìn trùng với hướng âm của trục Z. Trục X chỉ về phía phải và trục Y chỉ lên phía trên trong màn hình. Hệ tọa độ mới này sẽ được gọi là Hệ tọa độ Mắt (Eye Coordinate System). Phép biến đổi từ tọa độ thế giới sang các tọa độ mắt là một phép biến đổi affine, được gọi là phép biến đổi hiển thị (Viewing Transformation). Cả tọa độ thế giới và tọa độ mắt đều được biểu diễn bởi tọa độ đồng nhất (Homogeneous Coordinates) với w=1.

Bước thứ 2. Tọa độ mắt được chuyển qua tọa độ của thiết bị chuẩn hóa (Nomalized Device Coordinates) để cho vùng không gian mà ta muốn nhìn được đặt trong một khối lập phương tiêu chuẩn:

$$-1 \leq x \leq +1, -1 \leq y \leq +1, -1 \leq z \leq +1$$

Các điểm ở gần điểm nhìn (điểm đặt camera) hơn sẽ có thành phần z nhỏ hơn.

Bước cuối cùng, phép biến đổi cổng nhìn (Viewport Transformation) là sự kết hợp của 1 phép co giãn tuyến tính và 1 phép tịnh tiến. Sẽ chuyển thành phần x

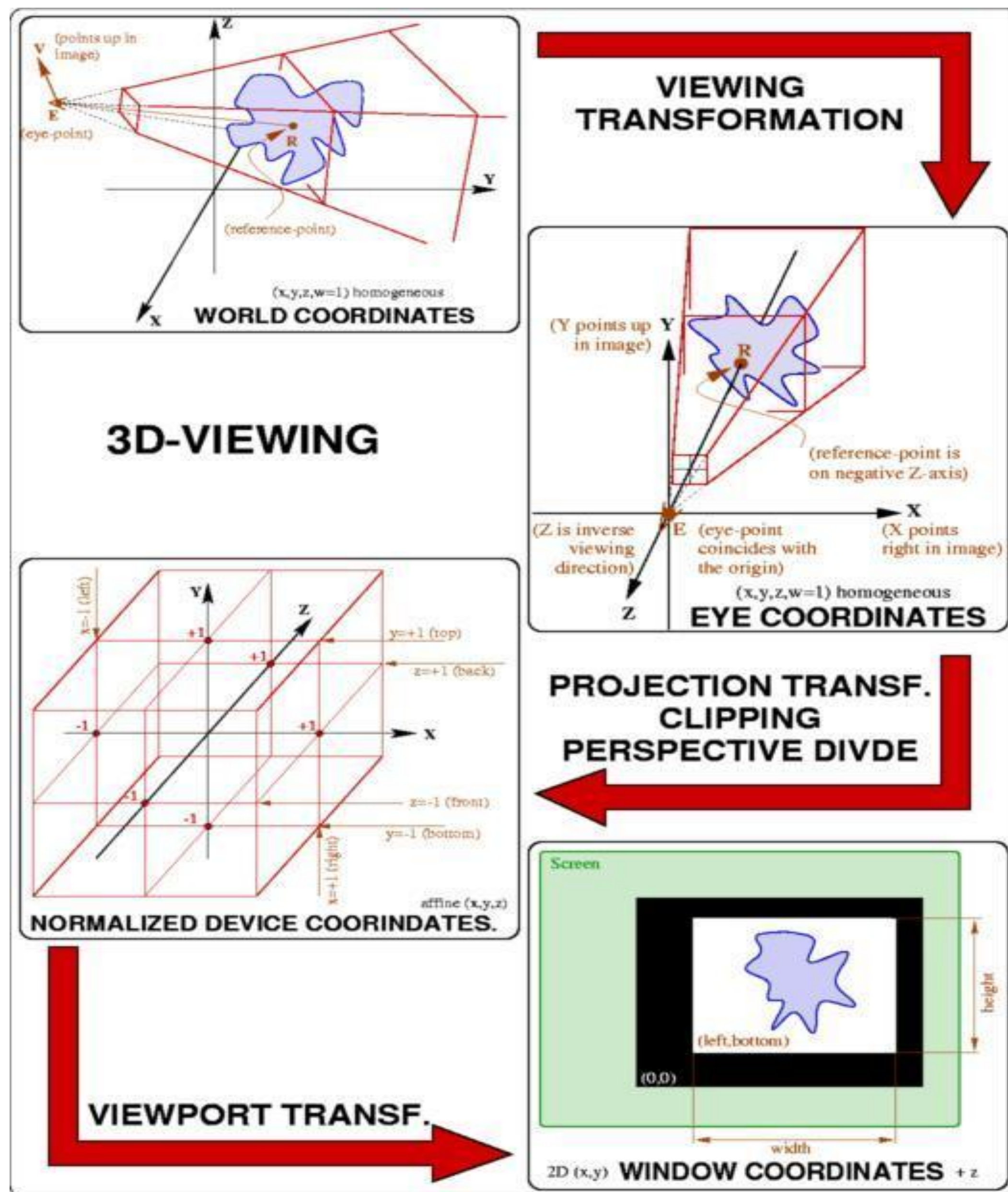
và y của tọa độ thiết bị chuẩn hóa $-1 \le x \le 1, -1 \le y \le 1$ sang tọa độ Pixel của màn hình. Thành phần z ($-1 \le z \le 1$) được chuyển sang đoạn [0,1] và sẽ được sử dụng như là giá trị chiều sâu (Depth-Value) trong thuật toán Z-Buffer (bộ đệm Z) được sử dụng cho việc xác định mặt sẽ được hiển thị.

Bước thứ 2 bao gồm 3 bước con :

- Một phép chiếu chuyển từ vùng nhìn sang 1 khối lập phương tiêu chuẩn với tọa độ đồng nhất: $-1 \le x \le 1, -1 \le y \le 1, -1 \le z \le 1$. Trong trường hợp sử dụng phép chiếu trực giao, vùng nhìn này sẽ có dạng một ống song song 3D với các mặt song song với các mặt của hệ tọa độ mắt. Trong trường hợp sử dụng phép chiếu đối xứng, vùng nhìn sẽ là một hình tháp cụt với đầu mút là gốc tọa độ của hệ tọa độ mắt. Hệ tọa độ đồng nhất (4 thành phần) thu được sau phép chiếu được gọi là hệ tọa độ cắt (Clipping Coordinate System). Phép chiếu sẽ là một phép biến đổi affine trong trường hợp phép chiếu là phép chiếu trực giao. Nếu phép chiếu là phép chiếu phối cảnh sẽ không phải là một phép biến đổi affine (Vì w sẽ nhận một giá trị khác 1)

Bước tiếp theo, các vùng của không gian hiển thị mà không nằm trong khối tiêu chuẩn đó (Khối này còn được gọi là khối nhìn tiêu chuẩn) sẽ bị cắt đi. Các đa giác, các đường thẳng được chứa trong hoặc là có một phần ở trong sẽ được thay đổi để chỉ phần nằm trong khối nhìn tiêu chuẩn mới được giữ lại. Phần còn lại không cần quan tâm nhiều nữa.

Sau khi cắt gọt, các tọa độ đồng nhất sẽ được chuyển sang tọa độ của thiết bị bằng cách chia x,y,z cho w. Nếu w nhận 1 giá trị đúng qua phép chiếu, thì phép chia này sẽ cho các động phối cảnh mong muốn trên màn hình. Vì lý do đó., phép chia này còn được gọi là phép chia phối cảnh (Perspective Division)



Hình 1.1: Tổng quan về hiển thị 3D và các phép chiếu.

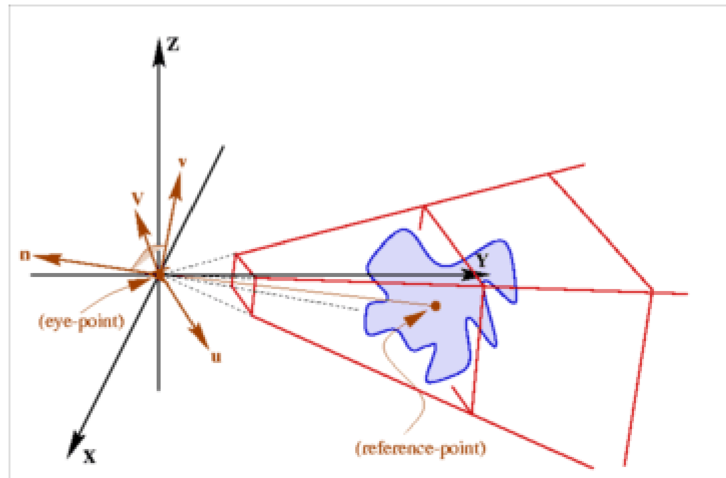
1.1.1.2 Phép biến đổi hiển thị (Viewing Transformation)

Phép biến đổi hiển thị sẽ đưa một camera ảo được cho tùy ý về một camera với điểm nhìn trùng với gốc tọa độ và hướng nhìn dọc theo chiều âm của trục Z (xem hình 2.1) Trục Y sau phép biến đổi tương ứng sẽ chỉ lên phía trên của màn hình. Trục X sẽ chỉ về phía phải.

Một cách thuận tiện để xác định vị trí của camera ảo là cho sẵn vị trí của điểm nhìn $\vec{E}$, Một điểm trong khung nhìn $\vec{R}$ (điểm tham chiếu) và một hướng $\vec{V}$ sẽ chỉ lên phía trên trong màn hình.

Phép biến đổi hiển thị sẽ gồm 2 bước:

- Một phép tịnh tiến sẽ đưa điểm nhìn $\vec{E}$ về gốc tọa độ. Ma trận biến đổi tương ứng sẽ là $M_t(-\vec{E})$. Kết quả sẽ như sau:



- Một phép quay sẽ chuyển hướng nhìn ngược về trục Z, quay vecto $\vec{V}$ về mặt phẳng YZ. Vector $\vec{V}$ sẽ chỉ được quay về trùng với trục Y nếu $\vec{V}$ vuông góc với hướng nhìn. Trước hết ta sẽ xây dựng tập các véc tơ chuẩn tắc phù hợp trong tọa độ thế giới.

$\vec{n} = \dfrac{\vec{E}-\vec{R}}{\left\|\vec{E}-\vec{R}\right\|}$ Ngược với hướng nhìn $\rightarrow \vec{Z} \quad (\overrightarrow{Oz})$

$\vec{u} = \dfrac{\vec{V}\times\vec{n}}{\left\|\vec{V}\times\vec{n}\right\|}$ Chỉ về phía phải, vuông góc với $\vec{n} \rightarrow \vec{X}$

$\vec{v} = \vec{n}\times\vec{u}$ Chỉ lên giống $\vec{V}$, nhưng vuông góc với $\vec{n}$ và $\vec{u} \rightarrow \vec{Y}$

Như vậy ma trận của phép quay sẽ là: $M_r(\vec{u},\vec{v},\vec{n})$

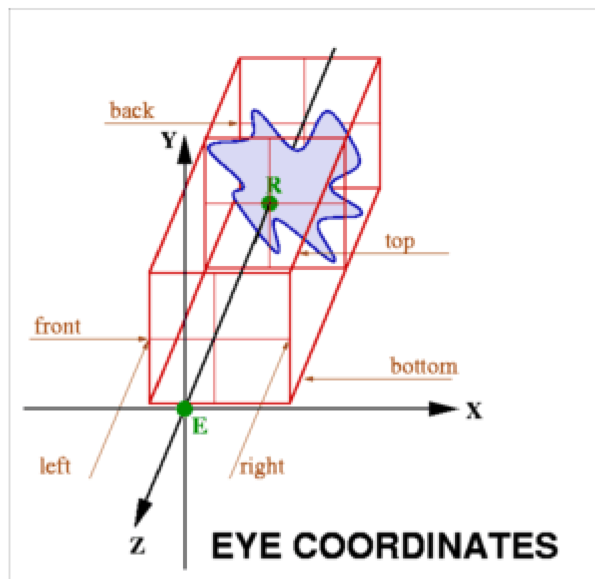
Và do đó ma trận của phép biến đổi sẽ là:

$$\mathrm{M_r}(\vec{u},\vec{v},\vec{n})\cdot\mathrm{M_t}(-\vec{E}) = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & -\vec{u}\cdot\vec{E} \\ v_x & v_y & v_z & -\vec{v}\cdot\vec{E} \\ n_x & n_y & n_z & -\vec{n}\cdot\vec{E} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Trong đó $\vec{u},\vec{v}$ và $\vec{v}$ được tính từ $\vec{E}$, $\vec{R}$ và $\vec{V}$

1.1.1.3 Phép chiếu trực giao (Orthographic Projection)

Trong trường hợp phép chiếu trực giao, vùng không gian hiển thị là một ống song song trong hệ tọa độ mắt. Các mặt của ống song song này song song với các mặt của hệ tọa độ mắt. Kích thước và vị trí của vùng không gian hiển thị được xác định bởi tọa độ mắt x_{left}, x_{right}, y_{bottom}, y_{top}, z_{front} và z_{back} . (x_{left}, y_{bottom}) và (x_{right}, y_{top}) xác định một cửa sổ trong mặt phẳng chiếu (hoặc là bất kỳ mặt nào song song với mặt XY) mà vùng không gian hiển thị sẽ được hiển thị trên đó. Cửa sổ này phải được đưa về dạng hình vuông $[-1,+1]^2$. z_{front} và z_{back} định nghĩa 2 mặt phẳng cắt trước và cắt sau. Tọa độ của tất cả các điểm trong không gian (hoặc ít nhất là những điểm ta muốn nhìn) phải thỏa mãn $z_{back} \leq z \leq z_{front}$. Khoảng giá trị của z phải được đưa về các giá trị chiều sâu (depth value) nằm trong đoạn [-1,+1]. Các điểm gần mắt hơn sẽ có giá trị chiều sâu nhỏ hơn.



Hình 1.2 : Vùng không gian hiển thị của phép chiếu trực giao.

Phép chiếu trực giao thu được bằng cách thực hiện các phép biến đổi sau theo thứ tự:

● Phép tịnh tiến $M_t(-\vec{M})$ sẽ đưa tâm của vùng không gian hiển thị về gốc tọa độ của hệ tọa độ mắt.

$$\vec{M} = \left(\frac{x_{\text{right}} + x_{\text{left}}}{2}, \frac{y_{\text{top}} + y_{\text{bottom}}}{2}, \frac{z_{\text{front}} + z_{\text{back}}}{2} \right)$$

● Một phép co giãn để đưa kích thước của vùng hiển thị về 2 đơn vị mỗi chiều.

● Một phép đối xứng qua mặt XY để các điểm nằm gần hơn sẽ nhận giá trị z nhỏ hơn.

Phép co giãn và phép đối xứng ở trên có thể thu được chỉ bằng một phép biến đổi đơn: $M_s(\vec{S})$ với:

$$\vec{S} = \left(\frac{2}{x_{\text{right}} - x_{\text{left}}}, \frac{2}{y_{\text{top}} - y_{\text{bottom}}}, \frac{-2}{z_{\text{front}} - z_{\text{back}}} \right)$$

Như vậy ma trận của phép chiếu trực giao sẽ là:

$$\mathbf{M_s}(\vec{S}) \cdot \mathbf{M_t}(-\vec{M}) = \begin{pmatrix} \frac{2}{x_{\text{right}} - x_{\text{left}}} & 0 & 0 & -\frac{x_{\text{right}} + x_{\text{left}}}{x_{\text{right}} - x_{\text{left}}} \\ 0 & \frac{2}{y_{\text{top}} - y_{\text{bottom}}} & 0 & -\frac{y_{\text{top}} + y_{\text{bottom}}}{y_{\text{top}} - y_{\text{bottom}}} \\ 0 & 0 & \frac{-2}{z_{\text{front}} - z_{\text{back}}} & \frac{z_{\text{front}} + z_{\text{back}}}{z_{\text{front}} - z_{\text{back}}} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Thành phần z không thay đổi, bởi vì phép chiếu trực giao là một phép biến đổi affine. Phép chiếu này được sử dụng trong các ứng dụng cần đến các quan hệ hình học (các tỉ số khoảng cách) như là trong CAD.

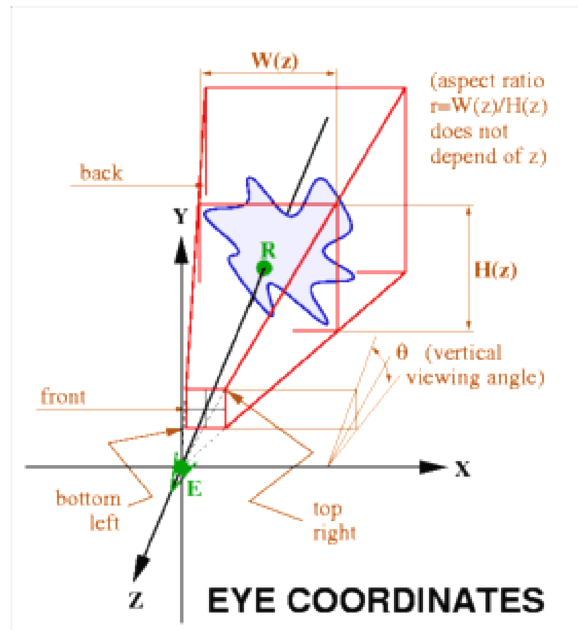
1.1.1.4 Phép chiếu phối cảnh (Perspective Projection)

Phép chiếu phối cảnh phù hợp và gần hơn với quan sát của con người (bằng một mắt) trong thế giới 3D. Tất cả các điểm trên một đường thẳng đi qua điểm nhìn sẽ được ánh xạ lên cùng một điểm trong màn hình 2D. Điểm ảnh này được xác định bởi tọa độ thiết bị chuẩn hóa x và y. Nếu 2 điểm được ánh xạ vào cùng một điểm trên màn hình, ta cần phải xác định điểm nào sẽ được hiển thị bằng thuật toán Z-buffer, nghĩa là so sánh chiều sâu của chúng. Vì lý do này chúng ta cần định nghĩa một thành phần tọa độ khác của thiết bị chuẩn hóa là z sao cho nó là một hàm tăng đơn điệu của khoảng cách từ điểm đó đến mặt phẳng mắt XY. Khoảng cách từ một điểm trong không gian đến mặt phẳng XY không bằng với khoảng cách từ điểm đó đến điểm nhìn (được đặt ở gốc tọa độ), nhưng nó sẽ được tính toán đơn giản hơn và cũng đủ để xác định được các mặt sẽ được hiển thị.

Như vậy, phép chiếu trực giao sẽ đưa một điểm (với tọa độ đồng nhất) trong hệ tọa độ mắt (x,y,z,1) về một điểm (tọa độ đồng nhất) trong hệ tọa độ cắt (x',y',z',w'). Sau đó các tọa độ của thiết bị chuẩn hóa (affine) (x'',y'',z'') sẽ thu được bằng cách chia x',y',z' cho w' (Phép chia phối cảnh):

$$(x'', y'', z'') = \left(\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right)$$

Với phép chiếu phối cảnh, vùng không gian hiển thị là một hình tháp cụt với đầu mút là gốc tọa độ.



Hình 1.3: Vùng không gian hiển thị của phép chiếu phối cảnh cân xứng (Symmetrical Perspective Projection)

1.1.1.5 Phép biến đổi cổng nhìn (Viewport Transformation)

Phép biến đổi cổng nhìn chỉ gồm một phép tịnh tiến và một phép thay đổi tỉ lệ để:

● Tọa độ thiết bị chuẩn hóa (x, y) với $-1 \le x \le 1, -1 \le y \le 1$ được chuyển qua tọa độ pixel.

$$left \le x_w \le left + width$$

$$bottom \le y_w \le bottom + height$$

● Thành phần z với $-1 \le z \le 1$ được co lại trong đoạn $0 \le z_w \le 1$.

Giá trị z_w này sẽ được sử dụng để loại bỏ những bề mặt bị ẩn. Những điểm có giá trị z_w nhỏ sẽ nằm trước những điểm có giá trị z_w lớn hơn.

Xây dựng ma trận biến đổi là công việc đơn giản. Tuy nhiên sẽ hiệu quả hơn nếu ta thực hiện phép biến đổi một cách trực tiếp:

$$x_w = left + width \cdot \frac{x+1}{2}$$

$$y_w = bottom + height \cdot \frac{y+1}{2}$$

$$z_w = \frac{z+1}{2}$$

1.1.2 Bộ đệm và các phép kiểm tra

Một mục đích quan trọng của hầu hết các chương trình đồ họa là vẽ được các bức tranh ra màn hình. Màn hình là một mảng hình vuông của các pixel. Mỗi pixel đó có thể hiển thị được 1 màu nhất định. Sau các quá trình quét (bao gồm Texturing và fog…), dữ liệu chưa trở thành pixel, nó vẫn chỉ là các "mảnh" (Fragments). Mỗi mảnh này chứa dữ liệu chung cho mỗi pixel bên trong nó như là màu sắc là giá trị chiều sâu. Các mảnh này sau đó sẽ qua một loạt các phép kiểm tra và các thao tác khác trước khi được vẽ ra màn hình.

Nếu mảnh đó qua được các phép kiểm tra (test pass) thì nó sẽ trở thành các pixel. Để vẽ các pixel này, ta cần phải biết được màu sắc của chúng là gì, và thông tin về màu sắc của mỗi pixel được lưu trong bộ đệm màu (Color Buffer).

Nơi lưu trữ dữ liệu cho từng pixel xuất hiện trên màn hình được gọi là *bộ đệm* (Buffer). Các bộ đệm khác nhau sẽ chưa một loại dữ liệu khác nhau cho pixel và bộ nhớ cho mỗi pixel có thể sẽ khác nhau giữa các bộ đệm. Nhưng trong một bộ đệm thì 2 pixel bất kỳ sẽ được cấp cùng một lượng bộ nhớ giống nhau. Một bộ đệm mà lưu trữ một bít thông tin cho mỗi pixel được gọi là một *bitplane*. Có các bộ đệm phổ biến như Color Buffer, Depth Buffer, Stencil Buffer, Accumulation Buffer.

1.1.2.1. Bộ đệm chiều sâu (Z-Buffer)

a. Khái niệm: Là bộ đệm lưu trữ giá trị chiều sâu cho từng Pixel. Nó được dùng trong việc loại bỏ các bề mặt ẩn. Giả sử 2 điểm sau các phép chiếu được ánh xạ vào cùng một pixel trên màn hình. Như vậy điểm nào có giá trị chiều sâu (z) nhỏ hơn sẽ được viết đè lên điểm có giá trị chiều sâu lớn hơn. Chính vì vậy nên ta gọi bộ đệm này là Z-buffer.

b. Depth test: Với mỗi pixel trên màn hình, bộ đệm chiều sâu lưu khoảng cách vuông góc từ điểm nhìn đến pixel đó. Nên nếu giá trị chiều sâu của một điểm được ánh xạ vào pixel đó nhỏ hơn giá trị được lưu trong bộ đêm chiều sâu thì điểm này được coi là qua Depth test (depth test pass) và giá trị chiều sâu của nó được

thay thế cho giá trị lưu trong bộ đệm. Nếu giá trị chiều sâu của điểm đó lớn hơn giá trị lưu trong Depth Buffer thì điểm đó "trượt" phép kiểm tra chiều sâu. (Depth test Fail)

1.1.2.2. Bộ đệm khuôn (Stencil Buffer)

a. Khái niệm: Bộ đệm khuôn dùng để giới hạn một vùng nhất định nào đó trong khung cảnh. Hay nói cách khác nó đánh dấu một vùng nào đó trên màn hình. Bộ đệm này được sử dụng để tạo ra bóng hoặc để tạo ra ảnh phản xạ của một vật thể qua gương…

b. Stencil Test: Phép kiểm tra Stencil chỉ được thực hiện khi có bộ đệm khuôn. (Nếu không có bộ đệm khuôn thì phép kiểm tra Stencil được coi là luôn pass). Phép kiểm tra Stencil sẽ so sánh giá trị lưu trong Stencil Buffer tại một Pixel với một giá trị tham chiếu theo một hàm so sánh cho trước nào đó. OpenGL cung cấp các hàm như là GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER hay là GL_NOTEQUAL. Giả sử hàm so sánh là GL_LESS, một "mảnh" (Fragments) được coi là qua phép kiểm tra (pass) nếu như giá trị tham chiếu nhỏ hơn giá trị lưu trong Stencil Buffer.

Ngoài ra OpenGL còn hỗ trợ một hàm là

glStencilOp(GLenum fail, GLenum zfail, GLenum zpass);

Hàm này xác định dữ liệu trong stencil Buffer sẽ thay đổi thế nào nếu như một "mảnh" pass hay fail phép kiểm tra stencil. 3 hàm fail, zfail và zpass có thể là GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR …Chúng tương ứng với giữ nguyên giá trị hiện tại, thay thế nó với 0, thay thế nó bởi một giá trị tham chiếu, tăng và giảm giá trị lưu trong stencil buffer. Hàm fail sẽ được sử dụng nếu như "mảnh" đó fail stencil test. Nếu nó pass thì hàm zfail sẽ được dùng nếu Depth test fail và tương tự, zpass được dùng nếu như Depth test pass hoặc nếu không có phép kiểm tra độ sâu nào được thực hiện. Mặc định cả 3 tham số này là GL_KEEP.

1.2 Bài toán tạo bóng

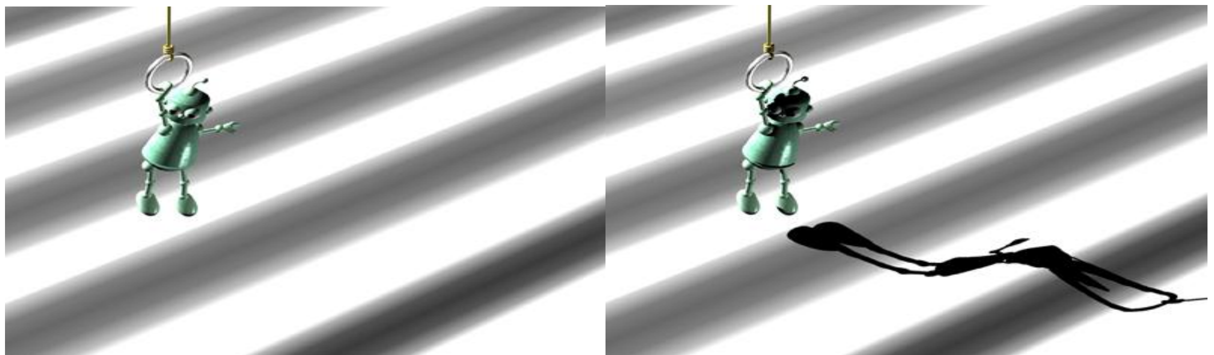
1.2.1 Bóng và các dạng nguồn sáng

1.2.1.1 Khái niệm bóng

"Bóng (Shadow) là một vùng tối nằm giữa một vùng được chiếu sáng, xuất hiện khi một vật thể được chiếu sáng toàn bộ hoặc một phần"

Bóng là một trong những yếu tố quan trọng nhất của tri giác con người về việc nhận biết các vật thể trong thế giới 3 chiều. Bóng giúp cho ta nhận biết được vị trí tương đối của vật đổ bóng (occluder) với mặt nhận bóng (receiver), nhận biết được kích thước và dạng hình học của cả vật đổ bóng và mặt nhận bóng.

Hình 1.4: Bóng cung cấp thông tin về vị trí tương đối của vật thể. Với ảnh ở bên trái ta không thể biết được vị trí của con rối. Nhưng với lần lượt 3 ảnh ở bên phải ta thấy vị khoảng cách của chúng so với mặt đất xa dần.



Hình 1.5: Bóng cung cấp thông tin về dạng hình học của mặt tiếp nhận. Hình bên trái ta không thể biết được dạng hình học của mặt tiếp nhận, còn mặt bên phải thì dễ dàng thấy được.

Hình 1.6: Bóng cung cấp thông tin về dạng hình học của con rối. Hình bên trái con rối cầm đồ chơi, ở giữa nó cầm cái vòng, và bên phải nó cầm cái ấm trà.

1.2.1.2 Phân loại bóng

Hầu hết các thuật toán và các phương pháp tạo bóng đều có thể được chia làm 2 loại chính là bóng cứng (Hard shadow) và bóng mềm (Soft shadow), phụ thuộc vào loại bóng mà nó tạo ra.

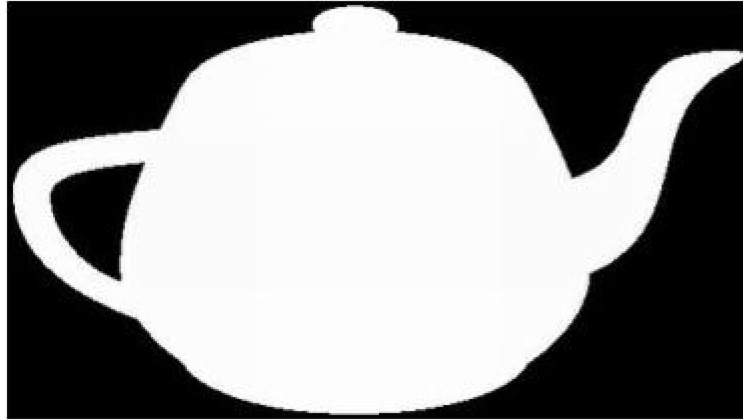
Vùng bóng được hiển thị được chia làm 2 phần phân biệt: Phần chính mà nằm hoàn toàn trong bóng được gọi là vùng thuần bóng, vùng bao bên ngoài nó và có một phần nằm trong bóng được gọi là vùng nửa bóng. Các thuật toán tạo bóng cứng là nhị phần vi mọi thứ đều chỉ có 2 trạng thái là bóng(1) và được chiếu sáng (0) – Chúng chỉ hiển thị duy nhất phần bóng của bóng. Các thuật toán tạo bóng mềm hiển thị vùng nửa bóng bên ngoài bao trùm vùng thuần bóng trung tâm và phải xử lý tính toán phần mờ đục cho vùng nửa bóng.(Kết quả từ sự phân bố cường độ ánh sáng bất quy tắc trong vùng nửa bóng)

Hình 1.7: Hình bên trái là một ví dụ về bóng cứng, hình bên phải là ví dụ về bóng mềm.

1.2.1.3 Các dạng nguồn sáng

Ánh sáng trong đồ họa 3D đóng vai trò khá quan trọng. Và đặc biệt nó là thành phần không thể thiếu để tạo ra bóng. Có nguồn sáng chỉ chiếu theo một hướng nhất đinh (giống ánh sáng mặt trời), có nguồn sáng chiếu ra toàn khung cảnh….Trong một khung cảnh có thể có nhiều nguồn sáng. Các nguồn sáng này có thể được tắt bật từng cái giống như ta tắt đèn bằng công tắc vậy. Theo mô hình ánh sáng của OpenGl thì ánh sáng gồm có 4 thành phần chính: Emissive Light, Ambient Light, Diffuse Light, Specular Light. Các thành phần này có thể được tính toán độc lập với nhau, và cuối cùng được kết hợp lại với nhau.

Ambient Light là ánh sáng bị phân rã bởi môi trường và không thể xác định hướng của chúng. Nếu trong một khung cảnh ta không xác định nguồn sáng thì kết quả đưa ra cũng giống như khi chúng ta sử dụng Ambient Light.



Hình 1.8: Chiếc ấm được chiếu bằng Ambient Light.

Diffuse Light (ánh sáng khuếch tán) là ánh sáng chiếu theo một hướng nhất, tuy nhiên khi nó gặp một bề mặt nó sẽ bị phân rã bằng nhau về mọi hướng, Vì thế nó sáng bằng nhau cho dù có đặt mắt nhìn ở đâu chăng nữa. Mọi nguồn sáng đến từ một điểm hay từ một hướng nhất định đều có thành phần Diffuse Light.

Hình 1.9: Ấm chè được chiều bằng Diffuse Light

Specular Light là ánh sáng phản xạ. Khi gặp một bề mặt nó sẽ phản xạ lại đúng theo quy luật phản xạ. Nó có thể được nhìn thấy trên những bề mặt cong.

Ánh sáng xung quanh 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. Thông thường ánh sáng xung quanh được xác định với một mức cụ thể gọi là mức sáng xung quanh của vùng không gian mà vật thể đó cư ngụ, sau đó ta cộng với cường độ sáng có được từ các nguồn sáng khác để có được cường độ sáng cuối cùng lên một điểm hay một mặt của vật thể

Nguồn sáng định hướng giống như những gì mà mặt trời cung cấp cho chúng ta. Nó bao gồm một tập các tia sáng song song, bất kể cường độ của chúng có giống

nhau hay không. Có hai loại kết quả của ánh sáng định hướng khi chúng chiếu đến bề mặt là: khuyếch tán và phản chiếu. Nếu bề mặt phản xạ toàn bộ (giống như mặt gương) thì các tia phản xạ sẽ có hướng ngược với hướng của góc tới (Hình 8.1). Trong trường hợp ngược lại, nếu bề mặt là không phản xạ toàn phần (có độ nhám, xù xì) thì một phần các tia sáng sẽ bị toả đi các hướng khác hay bị hấp thụ, phần còn lại thì phản xạ lại, và lượng ánh sáng phản xạ lại này tỷ lệ với góc tới. Ở đây chúng ta sẽ quan tâm đến hiện tượng phản xạ không toàn phần vì đây là hiện tượng phổ biến (vì chỉ có những đối tượng được cấu tạo từ những mặt như mặt gương mới xảy ra hiện tượng phản xạ toàn phần), và đồng thời tìm cách tính cường độ của ánh sáng phản xạ trên bề mặt.

1.2.2 Một số cách tiếp cận trong tạo bóng

1.2.2.1. Tạo bóng cứng

Các tính toán bóng thực chất là việc xác định xem một điểm trong khung nhìn có nằm trong vùng bóng không. Một cách cơ bản nó là một phép kiểm tra tính hiển thị của một điểm. Các thuật toán tạo bóng cứng phổ biến là:

- **Kĩ thuật tạo bóng giả (Fakes Shadow)** : các thuật toán tạo bóng giả bao gồm các trường hợp đặc biệt tạo bóng không đúng đắn bằng các phương pháp toán học. Những kĩ thuật này chỉ được sử dụng trong trường hợp đăc biệt(ví dụ như bóng chỉ vẽ cho những đối tượng đặc biệt, hay là bóng chỉ được vẽ lên một mặt phẳng). Tuy nhiên phương pháp này cũng tạo ra bóng làm cho ta có cảm giác khá thật.

- **Bóng khối (Shadow Volume)** : Bóng khối là kĩ thuật tạo bóng cần đến cấu trúc hình học của vật đổ bóng. Vật đổ bóng phải được tạo bởi các khối đa giác. Theo đó ta sẽ tìm các đỉnh và cạnh viền, là những cạnh đóng vai trò tạo nên bóng khối. Một tia sang khi chiếu tới vật thể sẽ tiếp xúc với vật thể tại điểm hoặc cạnh viền đó và đi cắt mặt phẳng nhận bóng. Những cạnh viền, đỉnh viền sẽ tạo ra các mặt bên đa giác của bóng khối. Từ đó dựa vào các phép kiểm tra ta sẽ kiểm tra được một điểm trong khung cảnh có thuộc bóng khối hay không.

- **Dùng bản đồ bóng (Shadow Mapping)** : Đây là thuật toán sử dụng đến bộ đệm chiều sâu (Depth Buffer) . Ý tưởng chủ yếu là sử dụng bản đồ chiều sâu (hay còn gọi là bản đồ bóng) để lưu trữ các giá trị chiều sâu khi tạo ảnh từ vị trí của ánh sáng rồi sau đó sử dụng các giá trị này để xác định pixel nào được chiếu sáng hay nằm trong bóng.

- **Lần theo tia sáng (Ray Tracing)** : với mỗi tia sáng đi ra từ mắt ta vào không gian là một đường thẳng sẽ cắt vào cửa sổ (màn hình) và chạm vào vật thể trong không gian (gần nhất từ mắt). Tại điểm chạm vào vật thể đó thì tùy mỗi điểm chạm ở vật thể đó có tính chất như thế nào mà ta chia ra các tia sáng khác nhau.

1.2.2.2. Tạo bóng mềm

Các kỹ thuật tạo bóng mềm sẽ cho bóng sinh ra trông thật hơn rất nhiều so với bóng được sinh ra bởi các thuật toán tạo bóng cứng. Tính thật của nó được biểu hiện bởi cả vùng nửa bóng và vùng thuần bóng. Hình dạng của bóng sinh ra bởi các thuật toán tạo bóng mềm sẽ phụ thuộc vào hình dạng, kích thước của vật thể tạo bóng , nguồn sáng và cả vị trí tương đối giữa nguồn sáng và vật thể.

Các kỹ thuật tạo bóng mềm phổ biến có thể kể đến là:

- **Thuật toán bộ đệm khung (Frame Buffer Algorithms):** Được đề xuất bởi Brotman và Badler dựa trên việc sinh ra các đa giác thuần bóng trong suốt quá trình tiền xử lý. Bộ đệm chiều sâu 2D mà được sử dụng để xác định mặt được hiển thị sẽ được mở rộng để lưu bộ đếm nắm giữ các thông tin để xác định xem một pixel bất kỳ là nằm trong vùng nửa bóng hay vùng thuần bóng.

- **Dõi quang tia 2 chiều và phân bố (Distributed and Bidirectional Ray Tracing) :**Rất nhiều mở rộng của thuật toán Ray-Tracing được sử dụng để tạo bóng mềm. Dõi quang tia phân bố cung cấp một kỹ thuật tạo bóng láng, mờ và chuyển động mờ trong khi Dõi quang tia 2 chiều cung cấp một phương pháp tạo bóng mềm rất nhanh.

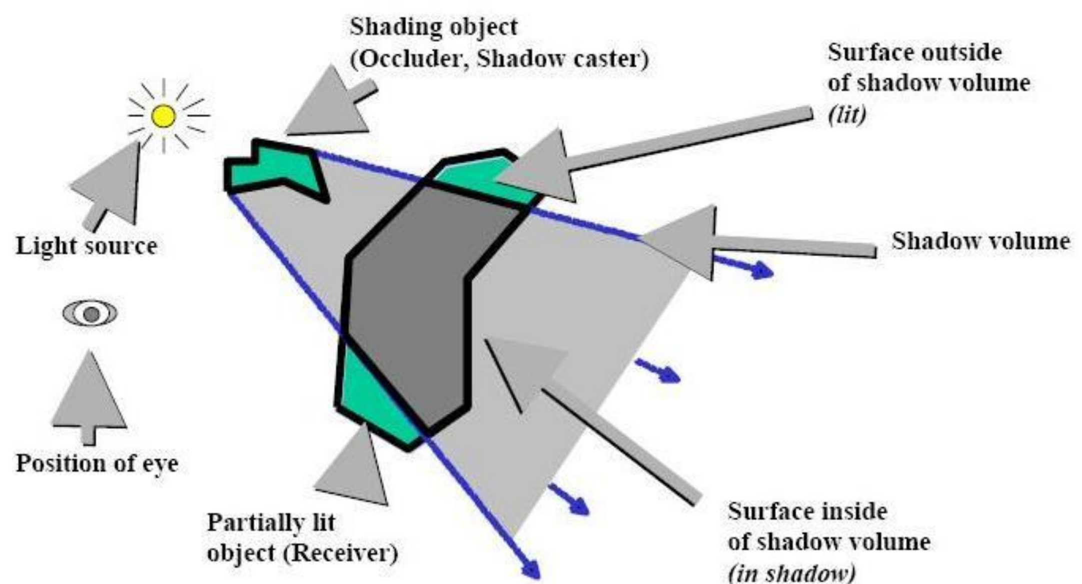
- **Ánh sáng nâng cao (Radiosity):**Radiosity là một kỹ thuật tạo bóng mềm bằng cách tính toán tất cả các phản xạ, khuếch tán ánh sáng giữa các mặt khác nhau của tất cả các vật thể trong khung cảnh. Nó hầu như chỉ được sử dụng cho các mặt đa giác bởi vì chi phí tính toán của phương pháp này rất lớn.

CHƯƠNG 2: KỸ THUẬT TẠO BÓNG CỨNG BẰNG PHƯƠNG PHÁP SHADOW VOLUME

2.1 Giới thiệu

Thuật toán tạo bóng bằng kỹ thuật sử dụng bóng khối được đề xuất đầu tiên bởi Frank Crow vào năm 1977. Theo đó ông ta vẽ một khối mà bị che lấp ánh sáng bởi vật tạo bóng. Mọi vật thể nằm trong khối đó được coi là nằm trong bóng.

"Bóng khối là một vùng không gian được kéo dài từ vật thể theo hướng của ánh sáng. Bất kỳ điểm nào nằm trong đó đều là thuộc vùng bóng của vật thể với ánh sáng"



Hình 2.1: Bóng khối

Thuật toán bóng khối là thuật toán tạo bóng dựa trên các thông tin về hình dạng của vật thể cần tạo bóng (Geometry Based Shadow Algorithm), vì thế nó đòi hỏi phải có các thông tin về tính kết nối của các lưới đa giác của tất cả các vật thể có trong khung hình (scene) để có thể tính toán một cách hiệu quả và chính xác.

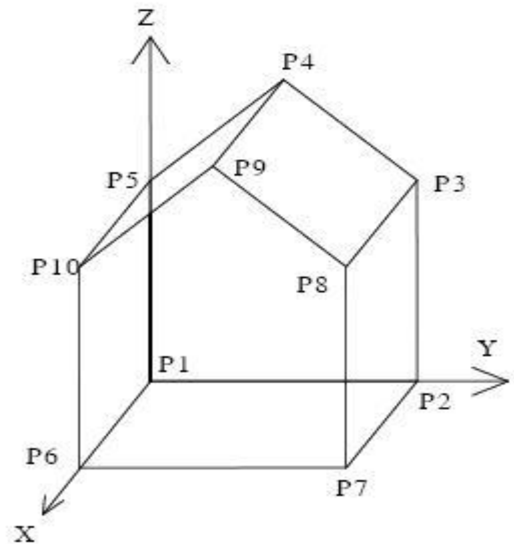
Các thông tin về vật thể có thể được lấy từ một mô hình WireFrame, trong đó nó thể hiện hình dạng của đối tượng 3D bằng 2 danh sách:

Danh sách các đỉnh: Lưu tọa độ các đỉnh.

Danh sách các cạnh: Lưu các cặp điểm đầu và cuối của từng cạnh.

Trong đó các đỉnh và các cạnh được đánh số thứ tự cho thích hợp.

Danh sách các đỉnh			
Vector	x	y	z
1	0	0	0
2	0	1	0
3	0	1	1
4	0	0.5	1.5
5	0	0	1
6	1	0	0
7	1	1	0
8	1	1	1
9	1	0.5	1.5
10	1	0	1



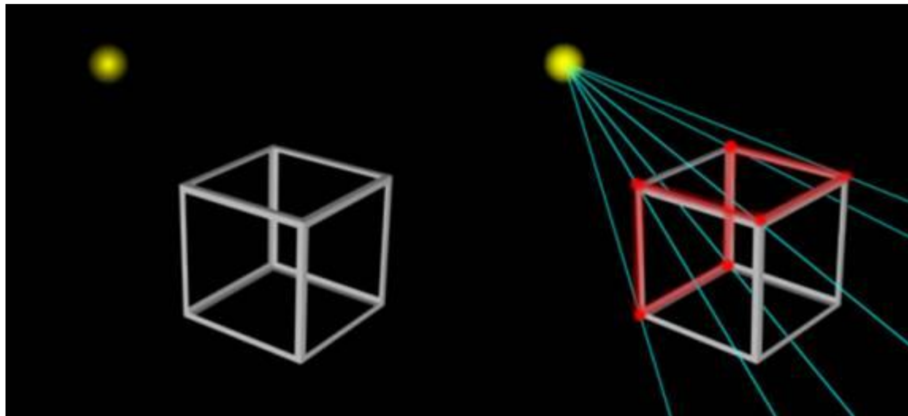
Hình 2.2: Biểu diễn của một căn nhà.

Ngoài ra còn có thể lấy từ file .obj được tạo ra khi ta sử dụng các công cụ xây dựng mô hình 3D như Google Sketchup, 3DSmax….

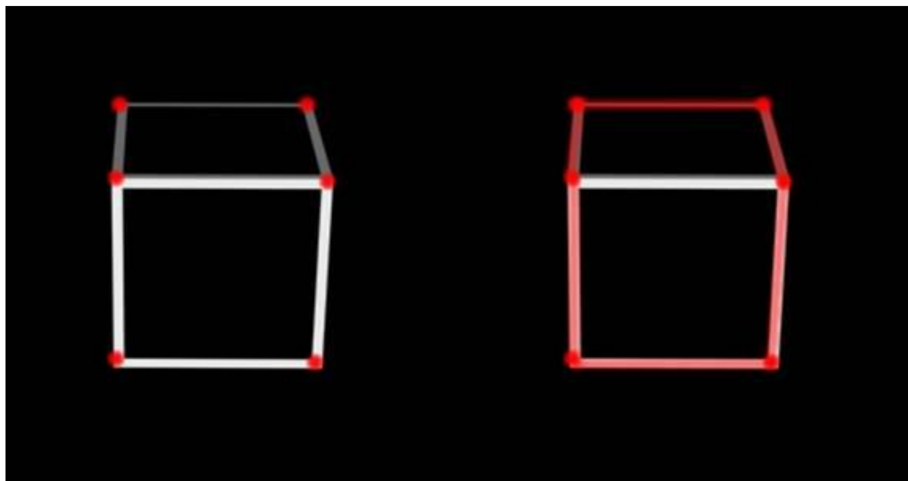
Thuật toán bóng khối còn là thuật toán trên từng pixel (Per Pixel Algorithm) Vì ta sẽ thực hiện một phép kiểm tra "trong bóng" (in shadow) cho mỗi điểm được vẽ ra màn hình. Nó bao gồm 2 phần riêng biệt. Phần đầu tiên chúng ta phải thực hiện các tính toán liên quan đến việc tạo ra cái mà người ta gọi là bóng khối.

2.2 Tìm danh sách cạnh viền.

Mỗi vật thể đối với mỗi nguồn sáng sẽ có một bóng khối. Để đơn giản ta sẽ chỉ xét với một nguồn sáng duy nhất. Ý tưởng để tạo ra bóng khối là ta sẽ xây dựng một lưới các đa giác bao quanh vùng bóng khối mà vật thể tạo ra do được chiếu sáng. Để làm được điều đó, ta phải tìm ra danh sách các cạnh viền của vật thể, chúng là những cạnh chủ yếu để tạo ra bóng khối, khi ánh sáng chiếu đến những cạnh đó, nó sẽ không dừng lại mà đi tiếp. Có thể hiểu đó là những cạnh tiếp xúc của vật thể với tia sáng.



Hình 2.3: Cạnh viền (Silhouette Edge) được tô đỏ.



Hình 2.4: Khi nhìn từ vị trí của nguồn sáng ta sẽ không thấy bóng và rất dễ để xác định cạnh và đỉnh viền.

Để tìm được các cạnh viền này, trước tiên ta cần xác định mặt nào của vật thể sẽ được chiếu bởi ánh sáng và mặt nào thì không. Việc này khá đơn giản khi ta đã biết tọa độ của nguồn sáng, vectơ pháp tuyến của mặt và phương trình của mặt phẳng. Ta chỉ việc thay tọa độ nguồn sáng vào phương trình mặt phẳng, rồi tính kết quả, nếu kết quả >0 thì khi đó pháp tuyến và nguồn sáng nằm cùng một phía với mặt phẳng đó, và do đó nó được chiếu sáng. Thủ tục xác định mặt được chiếu sáng sẽ như sau:

Gọi P[i](x,y,z) = a*x + b*y + c*z + d là phương trình của mặt thứ i của vật thể.

L = (L_x, L_y, L_z) là vị trí của nguồn sáng.

n: Số mặt của vật thể

Procedure VisiblePlaneTest( )

Begin

Side: ***interger***;

For i = 0 ***to*** n ***do***

Begin

Side =a*L_x + b*L_y + c*L_z + d;

if (Side>0) ***then*** P[i].visible = True

else P[i].visible = False;

End

End

Mỗi cạnh sẽ có 2 mặt chứa nó, mỗi cạnh viền sẽ phải có một đa giác kề được chiếu tới bởi ánh sáng và một thì bị che. Bởi vì nếu cả 2 đa giác đó đều được chiếu sáng hoặc là cả 2 đều bị che thì cạnh đó sẽ không phải là cạnh viền. Khi đó ta có thuật toán tìm danh sách các cạnh viền dưới dạng mã giả như sau:

Gọi P[i] là đa giác thứ i của vật thể.

n: là số đa giác.

Procedure Danhsachcanhvien()

Begin

for i = 0 ***to*** n ***do*** // Kiểm tra tất cả các đa giác.

if (P[i].visible = true} // Nếu mặt chứa đa giác đó được chiếu sáng

Begin

for {tất cả cạnh của đa giác} ***do***

if {cạnh đó đã có ở trong danh sách cạnh viền}

- Loại bỏ nó ra khỏi danh sách.

else

- Thêm cạnh đó vào danh sách.

End;

End;

Code tìm danh sách cạnh viền

```
for(int ring=1; ring<torusPrecision+1; ring++)
	{
		for(int i=0; i<torusPrecision+1; i++)
		{
vertices[ring*(torusPrecision+1)+i].position=vertices[i].position.GetRotated
Y(ring*360.0f/torusPrecision);
vertices[ring*(torusPrecision+1)+i].normal=vertices[i].normal.GetRotatedY(
ring*360.0f/torusPrecision);
		}
	}
	//calculate the indices
	for(int ring=0; ring<torusPrecision; ring++)
	{
		for(int i=0; i<torusPrecision; i++)
		{
indices[((ring*torusPrecision+i)*2)*3+0]=ring*(torusPrecision+1)+i;
indices[((ring*torusPrecision+i)*2)*3+1]=(ring+1)*(torusPrecision+1)+i;
indices[((ring*torusPrecision+i)*2)*3+2]=ring*(torusPrecision+1)+i+1;

indices[((ring*torusPrecision+i)*2+1)*3+0]=ring*(torusPrecision+1)+i+1;

indices[((ring*torusPrecision+i)*2+1)*3+1]=(ring+1)*(torusPrecision+1)+i;

indices[((ring*torusPrecision+i)*2+1)*3+2]=(ring+1)*(torusPrecision+1)+i+
1;	}
	}


	//Calculate the plane equation for each face
```

```
planeEquations=new PLANE[numTriangles];
if(!planeEquations)
{
    errorLog.OutputError("Unable to allocate memory for %d planes", numTriangles);
return false;
}
for(unsigned int j=0; j<numTriangles; ++j)
{
planeEquations[j].SetFromPoints(vertices[indices[j*3+0]].position,
vertices[indices[j*3+1]].position,
vertices[indices[j*3+2]].position);
}
```

2.3 Xác định các tứ giác bao quanh bóng khối

Khi chúng ta đã có danh sách các cạnh viền rồi, chúng ta sẽ tạo ra bóng khối bằng cách xây dựng các tứ giác từ mỗi cạnh viền đó dựa vào vị trí của nguồn sáng. 2 đỉnh đầu của tứ giác là 2 đỉnh của cạnh viền. 2 đỉnh tiếp theo sẽ nằm trên 2 đường thẳng nối giữa nguồn sáng và 2 đỉnh đầu. 2 đỉnh này theo lý thuyết sẽ được chiếu ra vô cực nhưng như thế sẽ không cần thiết vì thế ta sẽ chỉ cho chúng các giá trị tọa độ lớn là được.

v1 và v2 là 2 đỉnh của một cạnh viền bất kỳ trong danh sách.

L là vị trí của nguồn sáng.

v3 và v4 sẽ là 2 điểm cần tìm tọa độ để tạo ra tứ giác.

Const He_so_chieu 100 //Hệ số chiếu này phải là một số lớn.

v3.x = (v1.x - L.x) * He_so_chieu;

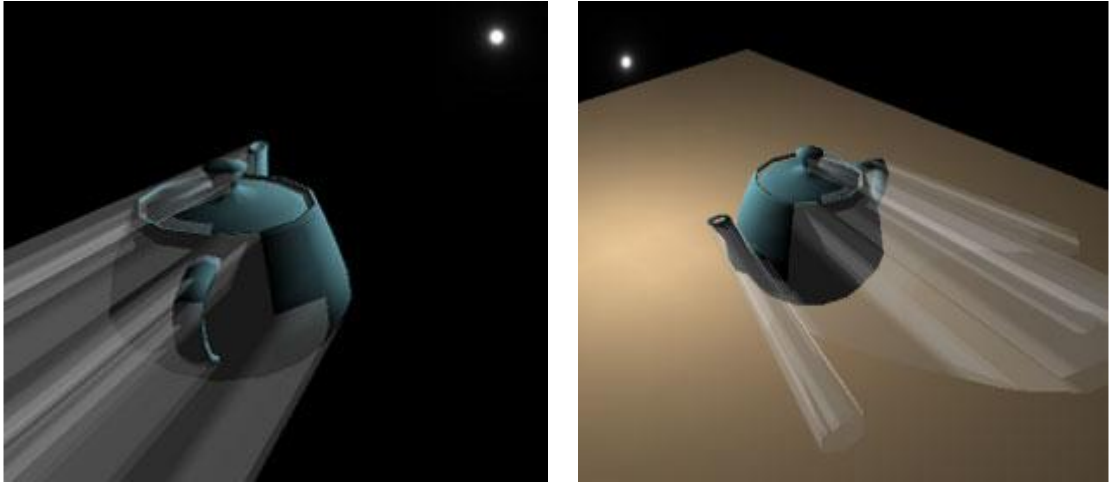
v3.y = (v1.y - L.y) * He_so_chieu;

v3.z = (v1.z - L.z) * He_so_chieu;

v4.x = (v2.x - L.x) * He_so_chieu;

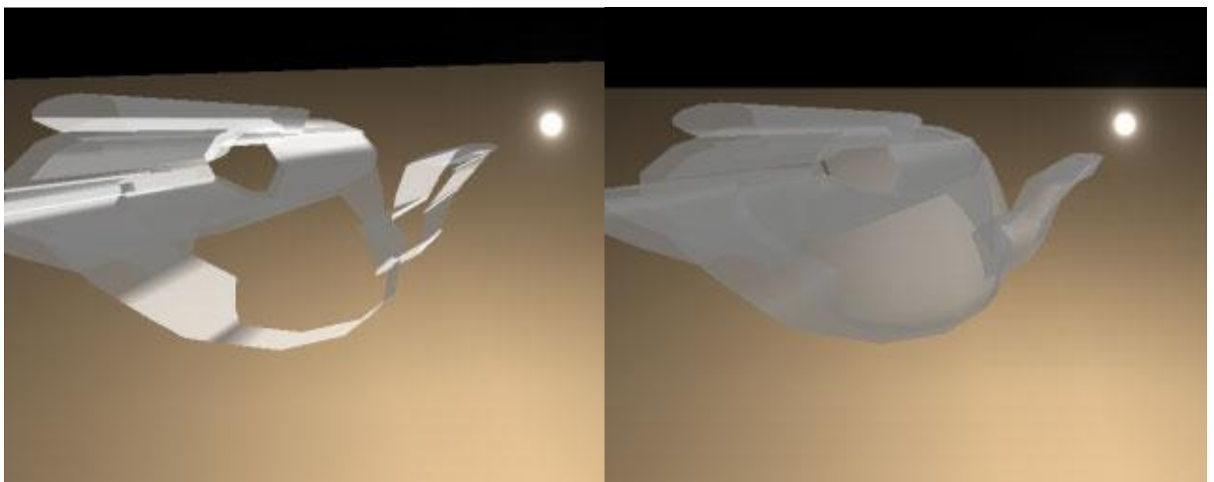
v4.y = (v2.y - L.y) * He_so_chieu;

v4.z = (v2.z - L.z) * He_so_chieu;



Hình 2.5: Bóng khối được tạo ra nhờ cạnh viền.

Từ các điểm này ta sẽ vẽ được các tứ giác bao ngoài bóng khối. Vấn đề còn lại cần phải giải quyết với bóng khối là phải "đậy nắp" (Capping) 2 đầu của khối lại để nó trở thành một khối kín. Lúc đó ta có thể thực hiện các phép kiểm tra một cách chính xác nhất. Để "nắp" phía trước thì đơn giản là ta dùng luôn các mặt trước của vật thể đối với vị trí của ánh sáng. Nắp mặt sau thì ta chỉ cần chiếu từng mặt sau của vật thể với ánh sáng đó ra vô cực. Phần này không cần thiết lắm bởi vì ta đã chiếu nó ra gần như là vô cực. Nên những điểm đó không cần xét đến nhiều.



Hình 2.6: Hình bên trái với bóng khối chưa được "đậy nắp", và hình bên phải là được "đậy nắp"

Code để xác định các tứ giác bao quanh bóng khối

```
isFacingLight=new bool[numTriangles];
    if(!isFacingLight)
    {
        errorLog.OutputError("Unable to allocate memory for %d booleans", numTriangles);
        return false;
    }
    //Create space for connectivity data
    neighbourIndices=new GLint[numTriangles*3];
    if(!neighbourIndices)
    {
        errorLog.OutputError("Unable to allocate memory for %d neighbour indices", numTriangles*3);
        return false;
    }
    //Create space for "is silhouette edge" booleans
    isSilhouetteEdge=new bool[numTriangles*3];
    if(!isSilhouetteEdge)
    {
        errorLog.OutputError("Unable to allocate memory for %d booleans", numTriangles*3);
        return false;
    }
    //Calculate the neighbours
    SetConnectivity();
    return true;
}
```

2.4 Tạo bóng khối bằng thuật toán Z-Pass.

Để xác định xem đoạn đó cắt bao nhiêu mặt trước, bao nhiêu mặt sau ta sẽ dùng một bộ đếm cho mỗi điểm cần kiểm tra, mà sẽ tăng lên 1 đơn vị khi nó đi xuyên qua một mặt trước và giảm đi một đơn vị nếu nó đi xuyên qua một mặt sau của bóng khối. Khi đó nếu bộ đếm cho giá trị bằng 0 thì điểm đó không nằm trong phần bóng, Còn nếu nó lớn hơn 0 thì có nghĩa là điểm này nằm trong vùng bóng và sẽ không được được vẽ ra.

Và Stencil Buffer sẽ thực hiện điều đó. Stencil Buffer sẽ cung cấp cho mỗi pixel trên màn hình một "bộ đếm" và chúng ta có thể tăng và giảm nó khi pixel đó được ghi vào trong Frame Buffer. Sau đó chúng ta hoàn toàn có thể kiểm tra bộ đếm đó để xác định xem điểm đó sẽ được ghi ra màn hình hay không.

Khi ta đã tạo được lưới các đa giác bao ngoài bóng khối. Chúng ta phải thực sự vẽ bóng của vật thể ra, hay nói chính xác là vẽ ra vật thể cùng với bóng của nó. Để làm được việc đó ta phải xác định được một pixel có nằm trong vùng bóng khối đó hay không. Thuật toán xác định một pixel có nằm trong vùng bóng khối đó hay không khá đơn giản. Tư tưởng của nó là, nối điểm cần kiểm tra với điểm đặt camera (mắt nhìn). Nếu số mặt trước và số mặt sau của bóng khối mà nó cắt bằng nhau thì điểm đó không nằm trong vùng bóng khối. Nếu nó cắt số mặt trước của bóng khối nhiều hơn số mặt sau mà nó cắt thì có nghĩa là điểm đó nằm trong vùng bóng khối.

Thuật toán sẽ được mô tả bằng mã giả như sau:

Procedure IN_SHADOW_TEST // Z-pass

For {tất cả các vật thể cần đổ bóng} ***do***

- Xây dựng danh sách các cạnh viền.

- Tính toán các tứ giác bao quanh bóng khối dựa trên các cạnh viền và từ vị trí của nguồn sáng.

End for

For {Tất cả các mặt trước của bóng khối nhìn từ vị trí của điểm nhìn} ***do***

if Depth test passes ***then***

- Tăng giá trị Stencil Buffer.

End if

End for

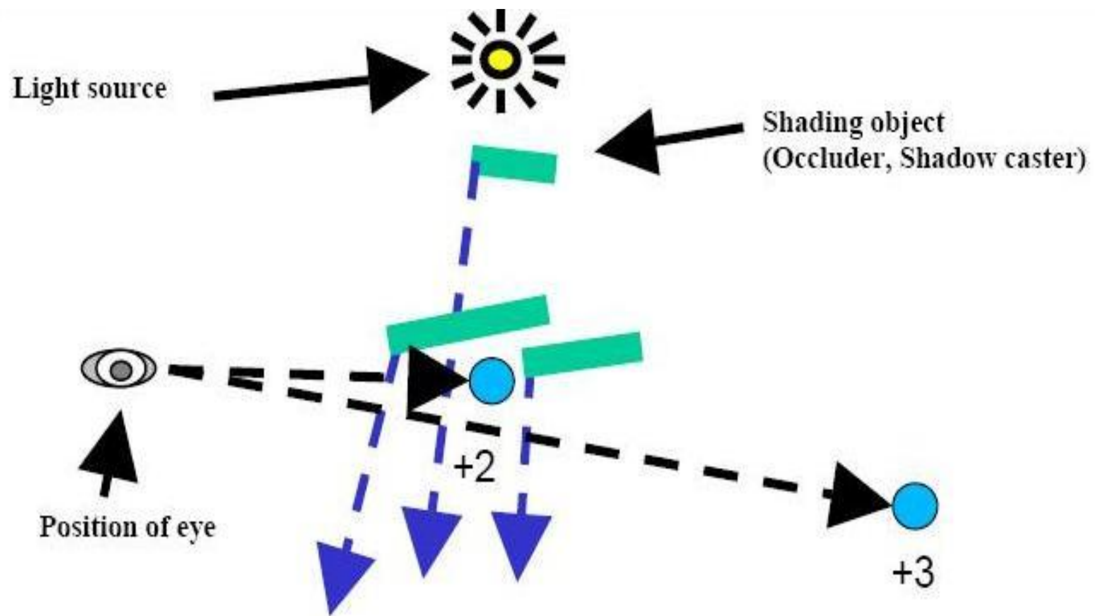
For {Tất cả các mặt sau của bóng khối nhìn từ vị trí của điểm nhìn} ***do***

if Depth test passes ***then***

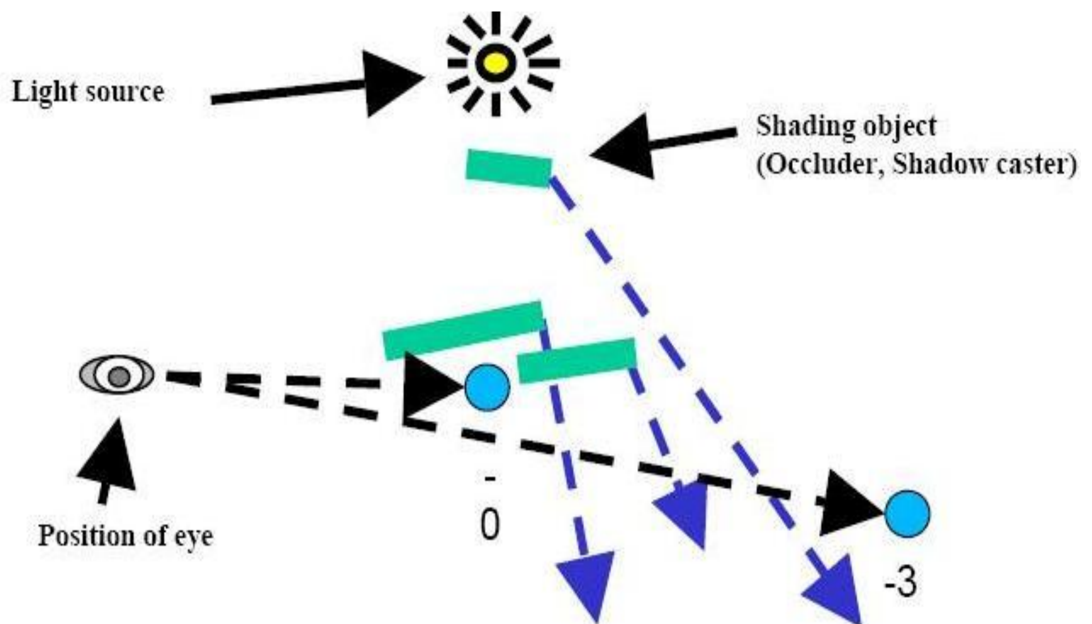
- Giảm giá trị Stencil Buffer.

End if

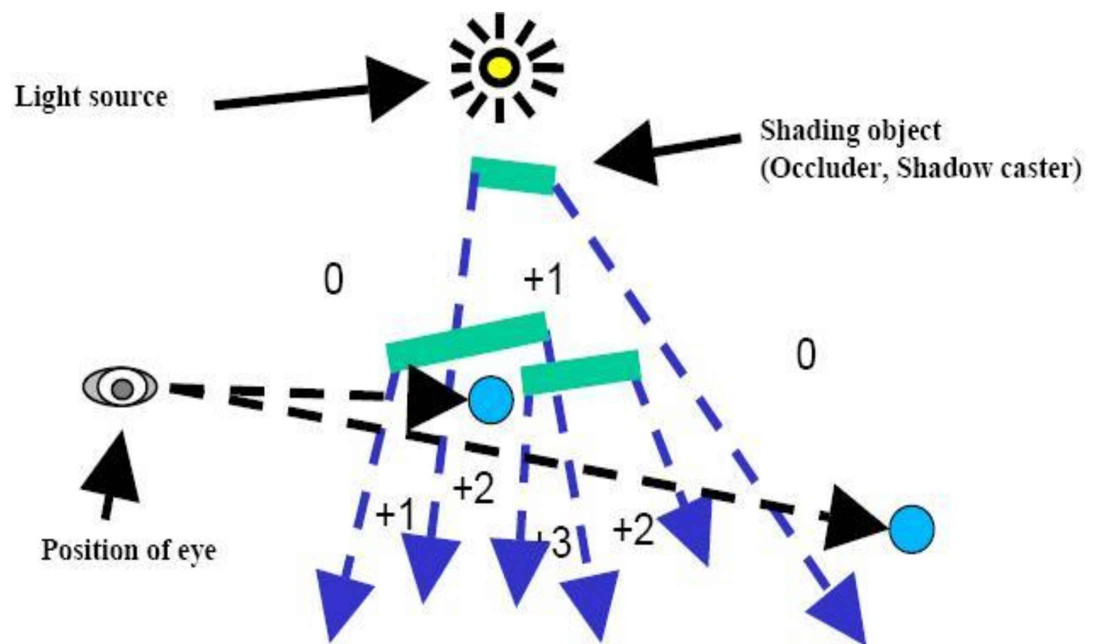
End for



Hình 2.7: Bước một, Vẽ các mặt trước của bóng khối.



Hình 2.8: Bước 2, Vẽ các mặt sau của bóng khối.



Hình 2.9: Kết quả. Giá trị Stencil Buffer tại các vùng.

Hình 2.10: Trái: Không có bóng, Giữa: Bóng khối được tạo ra, Phải: Kết quả cuối.

Các bước thực hiện như sau:

Xóa hết trong Z-buffer và Stencil-Buffer, Chắc chắn rằng Chế độ ghi vào Z-buffer và chế độ Stencil test được bật.

- Tạo ảnh của toàn bộ khung cảnh (bao gồm vật thể và các mặt hứng bóng) với Ambient Light để cho Z-buffer được cập nhật.
- Tắt chế độ ghi vào Z-buffer.
- Vẽ ra các mặt trước của bóng khối, Nếu chúng thực sự được vẽ ra. (Có nghĩa là Depth Pass) thì tăng giá trị Stencil Buffer.

- Vẽ các mặt sau của bóng khối, Nếu chúng thực sự được vẽ ra.(Có nghĩa là Depth Pass) thì giảm giá trị Stencil Buffer.

- Bật chế độ Stencil test (chỉ những điểm có giá trị Stencil = 0 mới được vẽ ra màn hình), Xóa Z-buffer, bật chế độ ghi vào Z-buffer, bật nguồn sáng.

- Vẽ ra toàn bộ khung cảnh những điểm có giá trị trong stencil Buffer là 0.

2.5 Tạo bóng bằng thuật toán Z-Fail

Thuật toán Z-Pass ở trên có một nhược điểm rất lớn là chưa xử lý được trường hợp khi điểm nhìn (viewpoint) nằm ở trong vùng bóng khối. Có 3 giải pháp để xử lý trường hợp này:

Trừ giá trị của Stencil Buffer 1 đơn vị cho phần bóng khối mà điểm nhìn nằm trong (trong trường hợp có nhiều vật thể). Tuy nhiên nếu làm như thế này thì chi phí tính toán sẽ rất đắt.

Tạo một mặt phẳng nằm trước và rất gần điểm nhìn cho mỗi phần bóng khối mà điểm nhìn nằm trong đó. Cách này cũng vậy, khá phức tạp và chi phí tính toán cũng đắt.

Cách thứ 3 là sử dụng thuật toán Z-Fail do. Thay vì tính toán giá trị Stencil bằng việc tăng các mặt trước của bóng khối và giảm giá trị của các mặt sau khi Z-Buffer Pass, toàn bộ quá trình sẽ được thay đổi để đếm từ vô cực thay vì đếm từ điểm nhìn. Vì thế thuật toán này còn gọi là Z-Fails.

Thuật toán Z-fail được thể hiện bằng đoạn mã giả sau:

Procedure IN_SHADOW_TEST // Z-fail

For {tất cả các vật thể cần đổ bóng} ***do***

- Xây dựng danh sách các cạnh viền.

- Tính toán các tứ giác bao quanh bóng khối dựa trên các cạnh viền và từ vị trí của nguồn sáng.

End for

For {Tất cả các mặt trước của bóng khối nhìn từ vị trí của điểm nhìn} ***do***

if Depth test fails ***then***

- Giảm giá trị Stencil Buffer.

End if

End for

For {Tất cả các mặt sau của bóng khối nhìn từ vị trí của điểm nhìn}

if Depth test fails ***then***

- Tăng giá trị Stencil Buffer.

End if

End for

2.5.1. Tất cả các mặt trước của bóng từ vị trí điểm nhìn

```
Void SHADOW_MODEL::SetConnectivity()
{
//set the neighbour indices to be -1
        for(unsigned int i=0; i<numTriangles*3; ++i)
                neighbourIndices[i]=-1;
//loop through triangles
        for(unsigned int i=0; i<numTriangles-1; ++i)
        {
//loop through edges on the first triangle
        for(int edgeI=0; edgeI<3; ++edgeI)
                {
//continue if this edge already has a neighbour set
                if(neighbourIndices[i*3+edgeI]!=-1)
                        continue;
//loop through triangles with greater indices than this one
        for(unsigned int j=i+1; j<numTriangles; ++j)
                {
                //loop through edges on triangle j
                for(int edgeJ=0; edgeJ<3; ++edgeJ)
                        {
```

```
        //get the vertex indices on each edge
        int edgeI1=indices[i*3+edgeI];
        int edgeI2=indices[i*3+(edgeI+1)%3];
        int edgeJ1=indices[j*3+edgeJ];
        int edgeJ2=indices[j*3+(edgeJ+1)%3];
    //if these are the same (possibly reversed order), these faces are neighbours
    if(  (edgeI1==edgeJ1 && edgeI2==edgeJ2)
        ||(edgeI1==edgeJ2 && edgeI2==edgeJ1))
        {       neighbourIndices[i*3+edgeI]=j;
        neighbourIndices[j*3+edgeJ]=i;
        }
```

2.5.2 Tất cả các mặt sau của bóng từ vị trí điểm nhìn

```
        VoidSHADOW_MODEL::CalculateSilhouetteEdges(VECTOR3D
lightPosition)
        {
                //Calculate which faces face the light
                for(unsigned int i=0; i<numTriangles; ++i)
                {
        if(planeEquations[i].ClassifyPoint(lightPosition)==POINT_IN_FRONT_OF
_PLANE)
                isFacingLight[i]=true;
                else
                isFacingLight[i]=false;
        }
                //loop through edges
                for(unsigned int i=0; i<numTriangles*3; ++i)
                {
                        //if this face is not facing the light, not a silhouette edge
```

```
                    if(!isFacingLight[i/3])
                    {
                              isSilhouetteEdge[i]=0;
                              continue;
                    }
//this face is facing the light
//if the neighbouring face is not facing the light, or there is no neighbouring face,
//then this is a silhouette edge
          if(neighbourIndices[i]==-1 || !isFacingLight[neighbourIndices[i]])
                    { isSilhouetteEdge[i]=1;
                         continue;
                    }
                              isSilhouetteEdge[i]=0;
                    }
          }
```

2.5.3. Vẽ bóng đậy nắp 2 đầu của khối

```
          Void  SHADOW_MODEL::DrawInfiniteShadowVolume(VECTOR3D
lightPosition, bool drawCaps)
{
glBegin(GL_QUADS);
          {
          for(unsigned int i=0; i<numTriangles; ++i)
          {
          //if this face does not face the light, continue
                    if(!isFacingLight[i])
                    continue;
          //Loop through edges on this face
                    for(int j=0; j<3; ++j)
```

```
                    {
    //Draw the shadow volume "edge" if this is a silhouette edge
    if(isSilhouetteEdge[i*3+j])
    {
    VECTOR3D vertex1=vertices[indices[i*3+j]].position;
    VECTOR3D vertex2=vertices[indices[i*3+(j+1)%3]].position;
    glVertex3fv(vertex2);
    glVertex3fv(vertex1);
    glVertex4f(vertex1.x-lightPosition.x,
        vertex1.y-lightPosition.y,
    vertex1.z-lightPosition.z, 0.0f);
    glVertex4f(   vertex2.x-lightPosition.x,
        vertex2.y-lightPosition.y,
    vertex2.z-lightPosition.z, 0.0f);
                        }
                    }
                }
            }
        glEnd();
//Draw caps if required
        if(drawCaps)
        {
            glBegin(GL_TRIANGLES);
            { for(unsigned int i=0; i<numTriangles; ++i)
                { for(int j=0; j<3; ++j)
                    {
VECTOR3D vertex=vertices[indices[i*3+j]].position;
```

```
        if(isFacingLight[i])
        glVertex3fv(vertex);
        else
        glVertex4f(    vertex.x-lightPosition.x,
    vertex.y-lightPosition.y,
    vertex.z-lightPosition.z, 0.0f);
                            }
                    }
                    glEnd();
            }
```

2.6 So sánh giữa 2 thuật toán

	Thuật toán Z-Pass	**Thuật toán Z-fail**
Ưu điểm	Không cần thiết phải "đậy nắp" (Cap) Tạo ít mặt hơn (do không cần tạo capping) Nhanh hơn Z-fail. Dễ thực hiện hơn	Giải quyết được trường hợp điểm nhìn nằm trong bóng khối.
Nhược điểm	Không giải quyết được vấn đề khi điểm nhìn ở trong bóng khối. Không có tự bóng (Self-shadow)	Chậm hơn Z-pass. Đòi hỏi bóng khối phải được Capping. Phải tạo ra nhiều mặt hơn do phải Capping. Khó thực hiện hơn. Không có tự bóng (Self-Shadow)

CHƯƠNG III: CHƯƠNG TRÌNH THỰC NGHIỆM

3.1 Bài toán

Chương trình ứng dụng kỹ thuật tạo bóng cứng minh họa cho kết quả đề tài sử dụng ngôn ngữ lập trình visual c.net và chạy trên môi trường window.Đầu vào của chương trình là 4 tori quay quanh 1 điểm sáng.

3.2 Phân tích, lựa chọn công cụ

3.2.1 Giới thiệu ngôn ngữ lập trình

Phần quan trọng của VS .NET là các công nghệ mới với trung tâm là .NET Framework - tập các tính năng Windows được xây dựng trên nền tảng môi trường thực thi ngôn ngữ chung (CLR – Common Language Runtime), trên đó là các lớp thư viện dùng để xây dựng ứng dụng Windows, ứng dụng web và dịch vụ web XML. Tất cả ngôn ngữ .NET đều được dịch sang dạng ngôn ngữ trung gian của Microsoft (MSIL – Microsoft Intermediate Language) trước rồi mới được dịch sang dạng mã thực thi bởi một trình dịch JIT (Just – in Time) trên nền .NET.

CLR và MSIL cho phép tất cả các ngôn ngữ .NET làm việc với nhau. Ví dụ bạn có thể dùng một lớp C# kế thừa từ một đối tượng COM C++/ATL, lớp này lại có thể bắt lỗi từ một chương trình Vbasic. Nhờ có sự hỗ trợ từ cấp hệ thống cho phép tích hợp đa ngôn ngữ, bạn có thể chạy từng bước để bẫy lỗi qua cả 3 ngôn ngữ trong cùng một môi trường phát triển ứng dụng VS.NET.

Trình dịch JIT cung cấp thêm khả năng bảo mật, tính an toàn lúc thực thi và khả năng chạy trên nhiều nền tảng (Microsoft cho biết sẽ dùng một tập con chuẩn hoá của .NET Framework - được gọi là nền tảng ngôn ngữ chung" - để xây dựng một thể hiện trên FreeBSD). Nếu trình JIT cho mã ngôn ngữ trung gian làm bạn liên tưởng đến Java thì cũng đừng hoang mang, Microsoft đã lẳng lặng dùng ngôn ngữ trung gian trong Vbasic và hầu hết các phần mềm ứng dụng của mình (kể cả Office)từ nhiều năm qua.

C# một ngôn ngữ phát triển từ C++ có nhiều nét giống Java, được đưa vào VS.NET là Visual C#.NET. Tương tự, ngôn ngữ Visual C++ quen thuộc được đóng gói vào VS.NET là Visual C++.NET bao gồm phần C++ truyền thống dùng cho ứng dụng độc lập với phần mở rông để dùng với nền .NET. Về phần VB.NET, đây là một cải tiến và không tương thích hoàn toàn với các phiên bản Vbasic. Tuy nhiên dường như những tính năng mới đã bù đắp khá nhiều cho sự không tương thích.

3.2.2 Lựa chọn công cụ

Mô tả: Dự án này sẽ hiển thị bốn Tori xoay quanh một điểm ánh sáng nguồn. Mỗi casts Shadows trên khác Tori và trên tường. Hiệu quả là đã đạt được trong ba qua:

1. Vẽ toàn bộ cảnh lit của ambient và một lượng nhỏ diffuse ánh sáng. Điều này cũng vượt qua đặt giá trị chiều sâu cho nhìn thấy cảnh.

2. Vẽ khối tin vào bóng tối stencil buffer. Theo mặc định, bóng tối khối tin, sử dụng "zFail" kỹ thuật, với một infinite clip bay xa, như được mô tả trong "mạnh mẽ Stenciled Shadow volumes" giấy.

3. Vẽ cảnh lit với đầy đủ diffuse và specular, trong khu vực unshadowed (nơi stencil == 0).

Yêu cầu:

Đối với hai bên-stencil Shadows:

EXT_stencil_two_side: khối lượng bóng tối có thể được rút ra chỉ một lần cho từng đối tượng, chứ không phải hai lần. Phần cứng sẽ tăng / giảm các bộ đệm stencil trên cơ sở đó chỉ đạo các đa giác đang phải đối mặt.

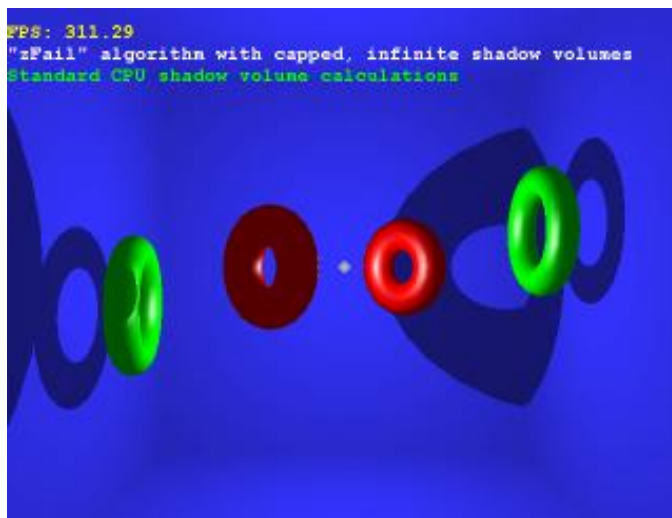
EXT_stencil_wrap

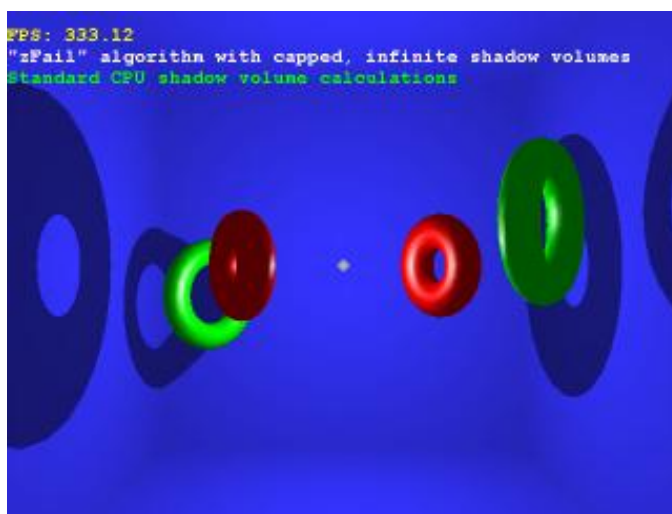
Đối với khối lượng chương trình vertex phun ra:

ARB_vertex_program: Một chương trình đơn giản tính toán đỉnh điểm giữa đỉnh bình thường và vector ánh sáng, và nếu điều này là tiêu cực, đỉnh được dự đi đến vô cùng. Do thiếu các vị trí bất biến giữa các chương trình đỉnh và các đường ống dẫn tiêu chuẩn, chương trình này để làm đường truyền ánh sáng.

3.3 Kết quả chương trình

- Click vào biểu tượng hình ảnh để chuyển hướng quan sát bóng.
- Khi các tori chạy quanh điểm sáng, bóng của nó sẽ song song được chiếu lên 4 bức tường xung quanh,
- Hình ảnh bóng rõ, cho ta cảm giác thật


FPS: 311.29
"zFail" algorithm with capped, infinite shadow volumes
Standard CPU shadow volume calculations


FPS: 333.12
"zFail" algorithm with capped, infinite shadow volumes
Standard CPU shadow volume calculations

KẾT LUẬN

Ngày nay với triển vượt bậc của công nghệ thông tin đặc biệt là đồ họa 3D đã cho ra đời những sản phẩm công nghệ có chất lượng,những hình ảnh sống động trong thế giới ảo.Và kỹ thuật tạo bóng cứng trong đồ họa 3D là một điển hình mà đại diện là kỹ thuật tạo bóng khối.Kỹ thuật này nhằm tạo nên khối lượng bóng khối bằng thuật toán z-fail,z-pass giúp cho hình ảnh của vật thể trở lên thật hơn,được ứng dụng rộng rãi trong game,phim hoạt hình,….

Nhận biết tầm quan trọng của việc ứng dụng bóng cứng trong đồ họa 3D, đồ án của em đã xây dựng thuật toán tạo bóng khối bằng thuật toán z-fail. Sau một quá trình nghiên cứu làm đồ án với sự chỉ dẫn nhiệt tình của thầy giáo hướng dẫn em đã học được cách tìm hiểu, phân tích và nghiên cứu một vấn đề khoa học mới. Trong thời gian làm đồ án tốt nghiệp, mặc dù bản thân đã rất nỗ lực, cố gắng, đầu tư nhiều thời gian, công sức cho việc tìm hiểu nghiên cứu đề tài và đã nhận được sự chỉ bảo, định hướng tận tình của thầy giáo hướng dẫn cùng các anh, chị đi trước nhưng do hạn chế về mặt thời gian và khó khăn trong việc tìm kiếm tài liệu, hạn chế về mặt kiến thức của bản thân, nên chưa có được kết quả thực sự hoàn hảo.Kính mong các thầy cô giáo cũng như các bạn chỉ bảo và giúp đỡ.

TÀI LIỆU THAM KHẢO:

Tiếng việt:

1. Đỗ Năng Toàn, Phạm Việt Bình (2008), Giáo trình Xử lý ảnh, Nhà xuất bản Khoa học và Kỹ thuật, Hà Nội.

2. Phạm Anh Phương, Nguyễn Hữu Tài, Giáo trình Lý thuyết Đồ họa, 15-09-2006

Tiếng anh:

3. Nehe tutorials bài học 28 - stencil bóng khối tin. Nehe.gamedev.net

4. Ikrima Elhassan, Shadow Algorithms,20-02-2007.

5. Practical & Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering, by Cass Everitt and Mark J Kilgard. developer.nvidia.com

6. Andrew V.Nealen, Shadow Volume and Shadow Mapping, Recent Development

7. In Real-time Shadow Rendering, University of British Columbia, 2000.