DÒ TIA TRONG ĐỒ HỌA MÁY TÍNH

MỤC TIÊU BUỔI HỌC

- DÒ TIA TRONG ĐỒ HỌA MÁY TÍNH
- CÁC ĐỐI TƯỢNG CƠ BẢN TRONG DÒ TIA
- TRIỂN KHAI DÒ TIA
- DÒ TIA THEO THỜI GIAN THỰC

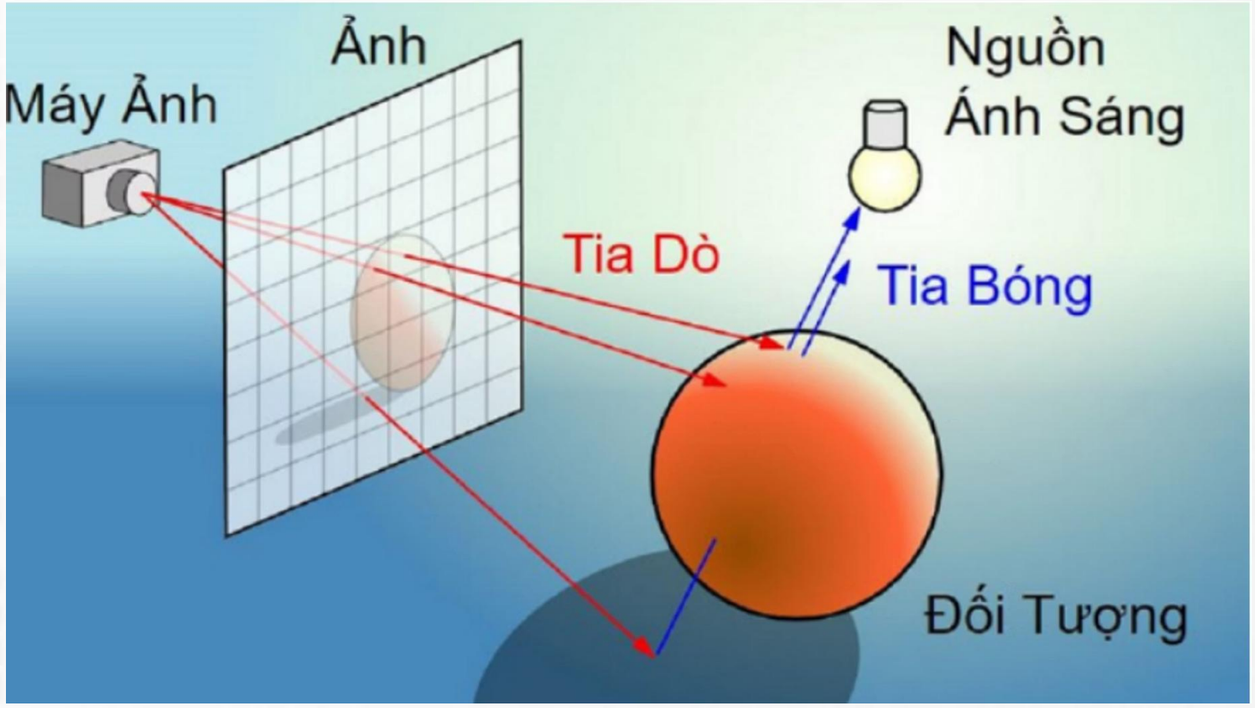
KHÁI NIỆM VỀ DÒ TIA (RAY TRACING)

1 ĐỊNH NGHĨA

Là kỹ thuật tạo hình ảnh kết xuất trên CPU bằng cách kiểm tra giao cắt giữa tia (ray) với các vật thể trong cảnh để tạo ra hình ảnh

2 CƠ CHẾ HOẠT ĐỘNG

- Khi một tia va chạm với vật thể, điểm giao cắt sẽ được xác định
- Nếu điểm đó nằm trong tầm nhìn, điểm ảnh (pixel) tương ứng sẽ được tô màu dựa trên vật liệu và thuộc tính của cảnh (ánh sáng, sương mù).



VAI TRÒ VÀ ỨNG DỤNG

TRONG ĐIỆN ẢNH

Sử dụng rộng rãi trong nhiều thập kỷ để sản xuất phim hoạt hình (ví dụ: Teenage Mutant Ninja Turtles).

TRONG LẬP TRÌNH GAME

Dù truyền thống là một kỹ thuật ngoại tuyến (offline), Ray Tracing vẫn có vai trò quan trọng trong phát triển game

Tiền xử lý (preprocessing): các hiệu ứng như tính toán ánh sáng và bóng đổ cố định, sau đó lưu vào kết cấu (texture) gọi là **Light maps**.

Ứng dụng phi đồ họa: Phát hiện va chạm (collision detection) hoặc trí tuệ nhân tạo (AI) để kiểm tra tầm nhìn (line-of-sight) giữa các nhân vật

Độ chân thực: Mô phỏng các hiện tượng vật lý phức tạp như phản xạ và khúc xạ.


RTX
ON
RTX
OFF

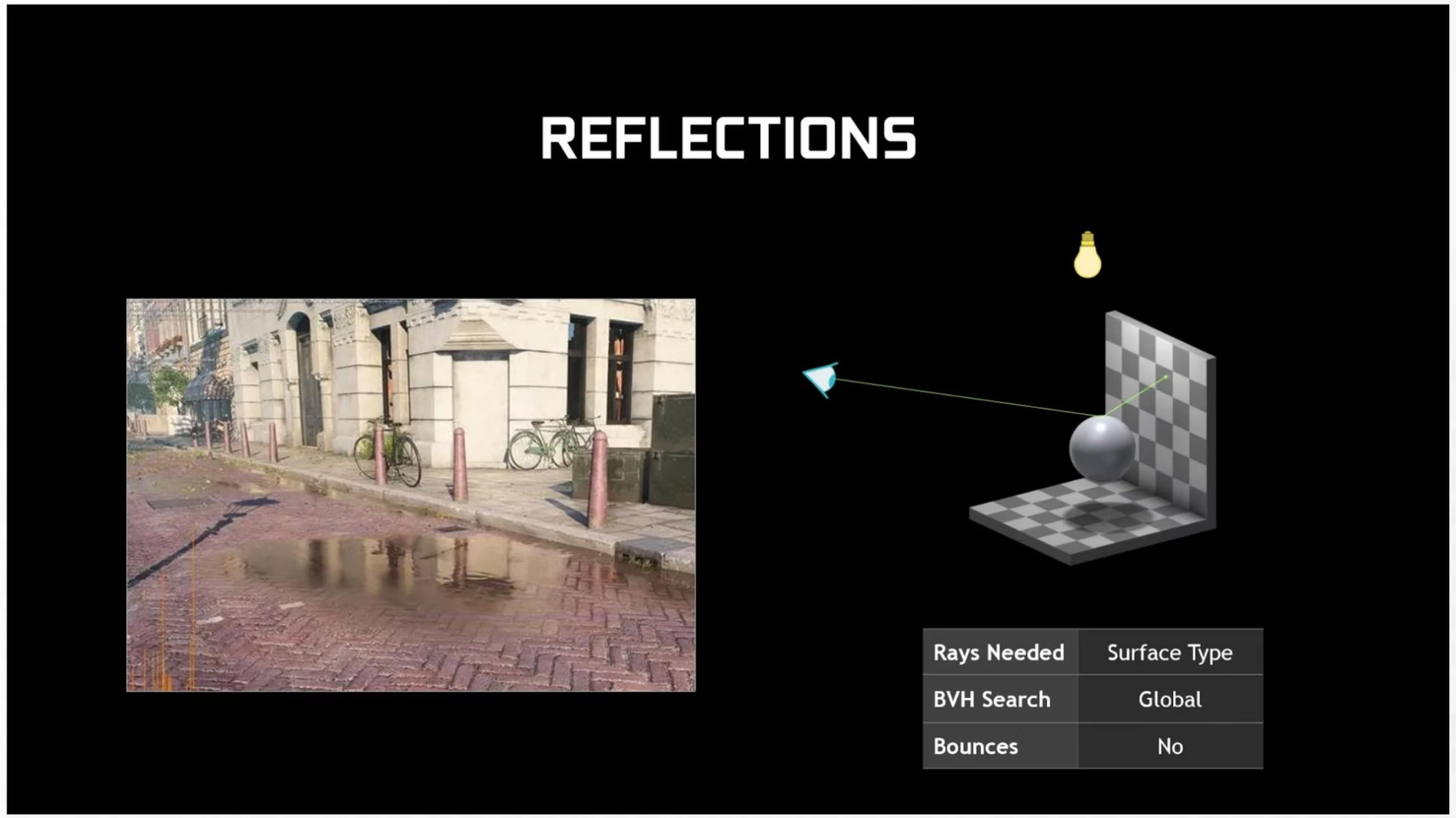
Lợi ích của Ray Tracing trong Game (PHẦN 1)

- PHẢN CHIẾU CHÂN THỰC HƠN
- BÓNG ĐỔ MỀM MẠI VÀ CHÍNH XÁC
- ÁNH SÁNG TỰ NHIÊN VÀ SINH ĐỘNG
- TĂNG CẢM GIÁC NHẬP VAI
- HỖ TRỢ TRONG CHƠI GAME
- GIÚP LÀM MOD ĐẸP HƠN
- GIẢM DUNG LƯỢNG GAME
- TIẾT KIỆM THỜI GIAN PHÁT TRIỂN


OKIE GRILL
& SCOPHOUSE
RTX
ON


OKIE GRILL
& SCOPHOUSE
RTX
OFF

ĐẶC ĐIỂM - PHẢN CHIẾU (REFLECTION)



NGUYÊN LÝ

Hệ thống dò theo tia sáng từ nguồn đánh lên bề mặt vật liệu.

KẾT QUẢ

Tạo ra hình ảnh phản chiếu theo thời gian thực, chuyển động khớp với môi trường bên ngoài.

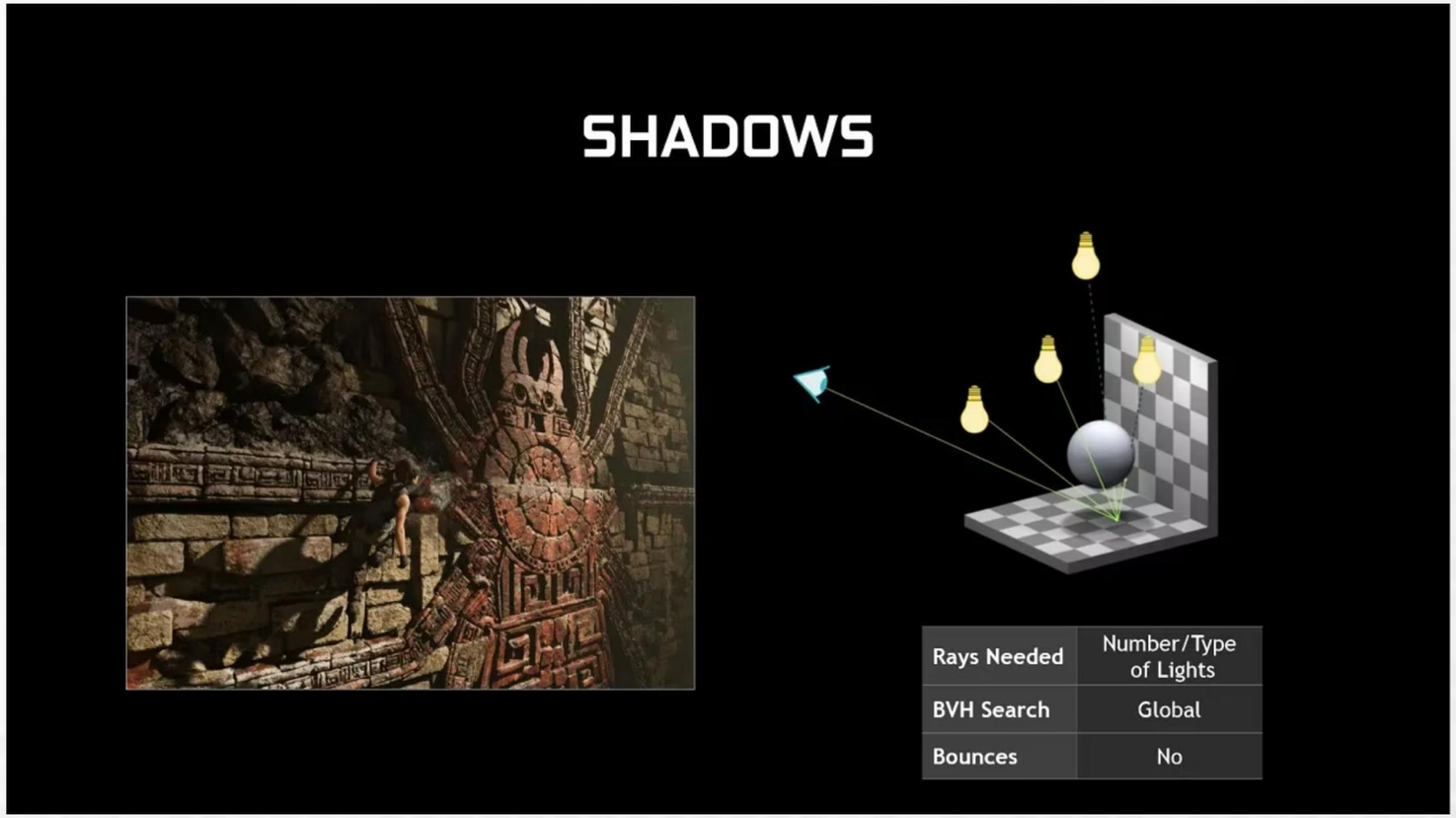
ĐẶC ĐIỂM - ĐỔ BÓNG (RAY TRACED SHADOWS)

NGUYÊN LÝ

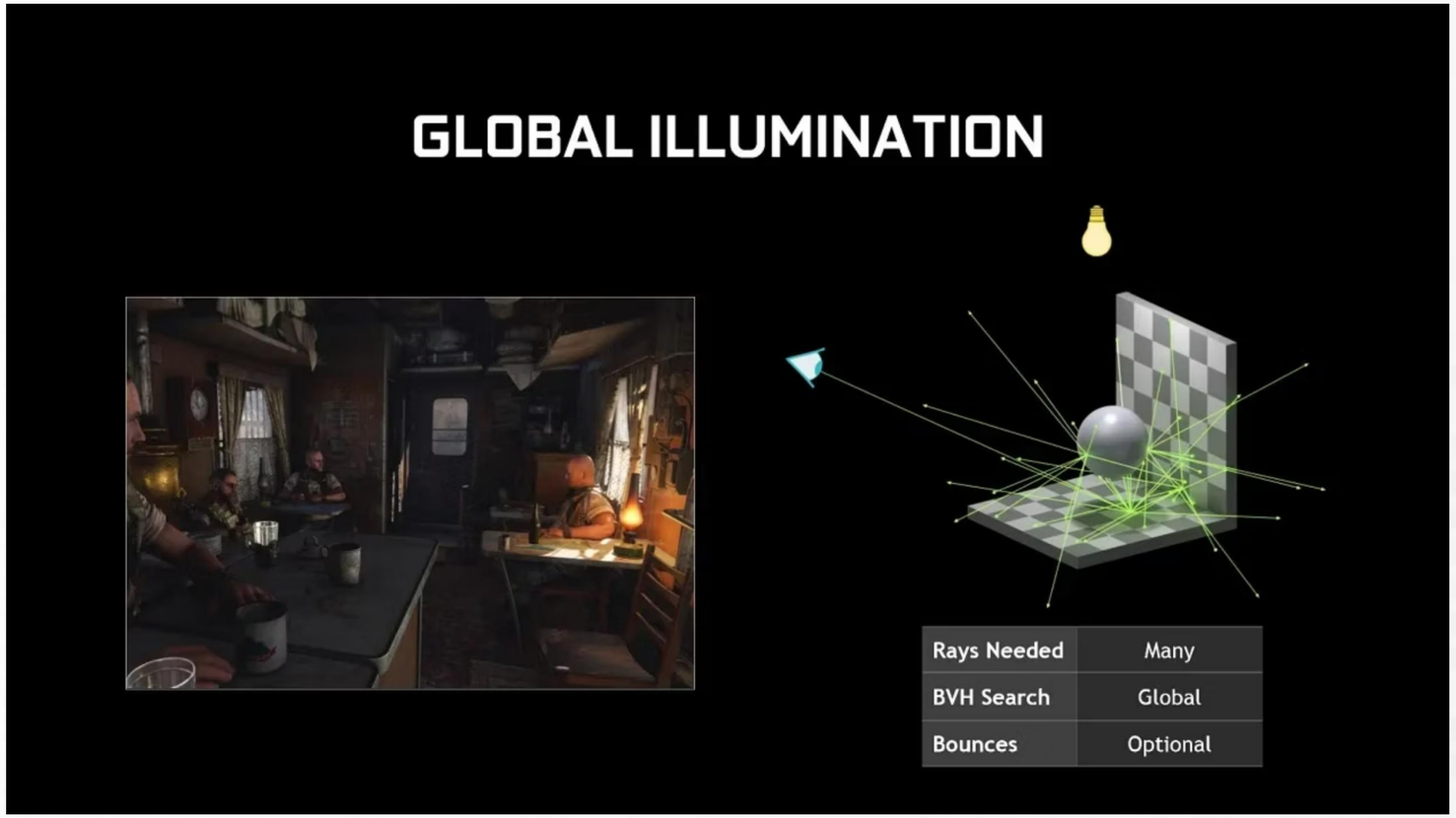
Các nhân xử lý dò tất cả nguồn sáng để xác định độ che khuất của vật thể.

ĐẶC ĐIỂM

Phần bóng tối có độ trong trẻo, màu sắc và hướng tuân thủ các định luật vật lý tự nhiên.



ĐẶC ĐIỂM - CHIẾU SÁNG TỔNG THỂ (GLOBAL ILLUMINATION)



VAI TRÒ

Thể hiện khả năng phản xạ ánh sáng của bề mặt vật liệu ra môi trường xung quanh.

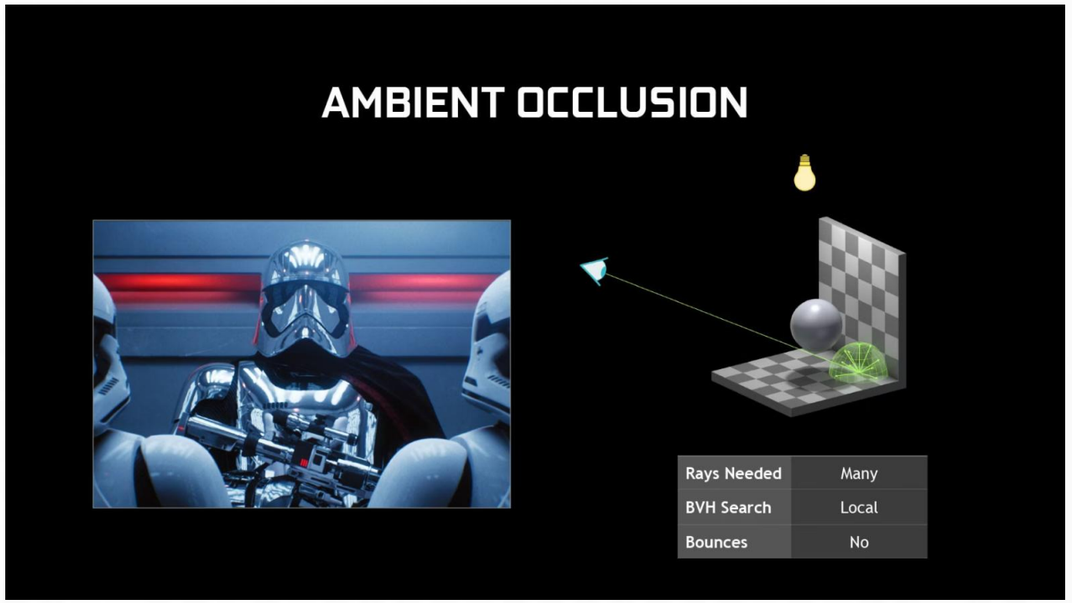
YÊU CẦU

Đòi hỏi phần cứng khổng lồ để xử lý lượng lớn tia sáng phản xạ và tán xạ hỗn loạn.

ĐẶC ĐIỂM - ĐỔ BÓNG MÔI TRƯỜNG & PHÁT XẠ

ĐỔ BÓNG MÔI TRƯỜNG (AMBIENT OCCLUSION)

Thuật toán tính toán phơi sáng và che tối riêng biệt để tạo độ nổi khối cho vật thể.



PHÁT XẠ ÁNH SÁNG (EMISSIVE LIGHTING)

Giả lập luồng sáng nhỏ với cường độ khác nhau, tương tác chân thực với môi trường.

MỘT SỐ TỰA GAME HỖ TRỢ RAYTRACING

RAY TRACING TRÊN CÁC THIẾT BỊ

MÁY TÍNH/LAPTOP

Sử dụng card đồ họa rời với các lõi chuyên dụng như NVIDIA RT Cores hoặc AMD Ray Accelerators.

ĐIỆN THOẠI THÔNG MINH

Bắt đầu hỗ trợ trên các dòng cao cấp thông qua chip SoC.

Hạn chế: Hiệu quả thấp hơn PC, gây ngốn pin nhanh và tỏa nhiệt.

RAY TRACING: ĐIỆN THOẠI VS MÁY TÍNH

SỨC MẠNH XỬ LÝ

Do hạn chế về GPU, smartphone thường chỉ xử lý các hiệu ứng Ray Tracing cơ bản như phản xạ hoặc đổ bóng.

TỐI ƯU PHẦN MỀM

Áp dụng kỹ thuật Ray Tracing lai (hybrid) kết hợp với rasterization để đạt hiệu quả tối ưu.

CHẤT LƯỢNG HIỂN THỊ

Thường chạy ở khung hình hoặc độ phân giải thấp hơn nhằm giảm tải cho chip xử lý.

KỸ THUẬT DÒ TIA TIẾN (FORWARD RAY TRACING)

NGUYÊN LÝ

Tia được bắn ra từ nguồn sáng vào cảnh 3D. Pixel được tô màu khi tia chạm vật thể trong tầm nhìn, pixel tương ứng sẽ được tô màu

NHƯỢC ĐIỂM

Rất kém hiệu quả vì lãng phí tài nguyên cho các tia không bao giờ đi tới ống kính hoặc không va chạm với vật thể trong tầm nhìn ("wasted rays").

A few rays being traced from a light source into a scene (left). Many rays being traced from a light source (right).

CÁC BƯỚC TẠO MỘT BỘ DÒ TIA TIẾN ĐƠN GIẢN

THIẾT LẬP CẢNH

Xác định vị trí các hình cầu và nguồn sáng.

PHÁT TIA

Từ mỗi nguồn sáng, phát ra một lượng hữu hạn tia vào cảnh.

KIỂM TRA GIAO CẮT

Xác định điểm giao cắt và kiểm tra xem chúng có nằm trong thể tích nhìn hay không.

TÔ MÀU PIXEL

Với mỗi giao cắt hợp lệ trong vùng nhìn, tô màu pixel tương ứng cdựa trên thuộc tính vật liệu, nguồn sáng và toàn bộ cảnh.

KẾT XUẤT & HIỂN THỊ

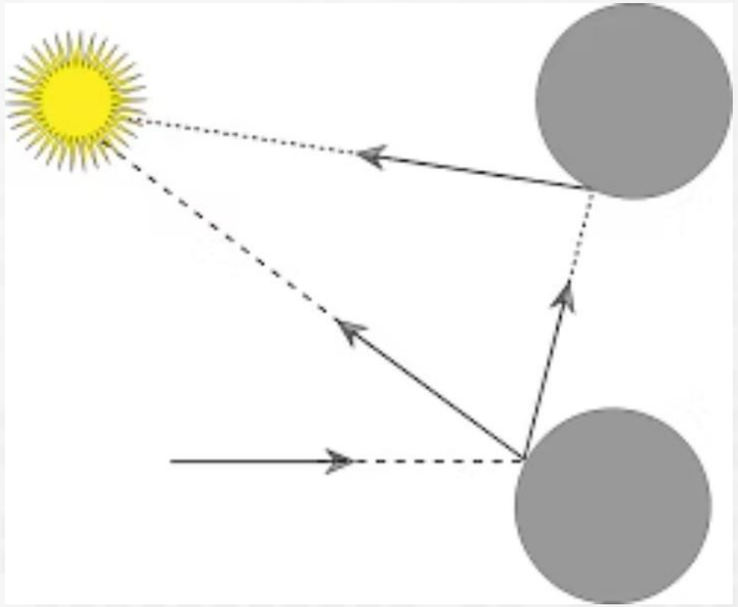
Lưu hình ảnh đã kết xuất ra file hoặc hiển thị lên màn hình.

Lưu ý: Số lượng tia càng nhiều, chất lượng hình ảnh càng cao nhưng chi phí hiệu năng cũng lớn hơn đáng kể.

```
function Forward_RayTracing(scene)
{
   foreach(light in scene)
   {
      foreach(ray in light)
      {
         closestObject = null;
         minDist = “Some Large Value”;
          foreach(object in scene)
          {
             if(ray->Intersects(object) && object->distance <
minDist)
             {
                minDist = object->distance;
                point_of_intersect = ray->Get_Intersection();
                closestObject = object;
             }
          }

          if(Is_In_View(point_of_intersect) && closestObject !=
NULL)
          {
             Shade_Pixel(point_of_intersect, light,
closestObject);
          }
       }
    }

    Save_Render("output.jpg");
 }
```

KẾT LUẬN

- Rất kém hiệu quả vì phần lớn các tia từ nguồn sáng không bao giờ đi tới ống kính camera hoặc không va chạm với vật thể nằm trong tầm nhìn ==> lãng phí tài nguyên tính toán (gọi là "wasted rays").
- Chất lượng của kết quả hiển thị phụ thuộc rất nhiều vào số lượng tia được dò trong cảnh, nhiều tia trong số đó có thể không ảnh hưởng đến cảnh được hiển thị ==> Nếu không sử dụng đủ tia hoặc nếu chúng không được phân bổ hiệu quả, một số pixel có thể không được tô bóng, mặc dù chúng đáng lẽ phải được tô bóng.
- Không thể thực hiện một cách hợp lý trong thời gian thực.
- Nó thường lỗi thời đối với đồ họa máy tính và trò chơi.
- Việc tối ưu hóa có thể khó khăn hơn.



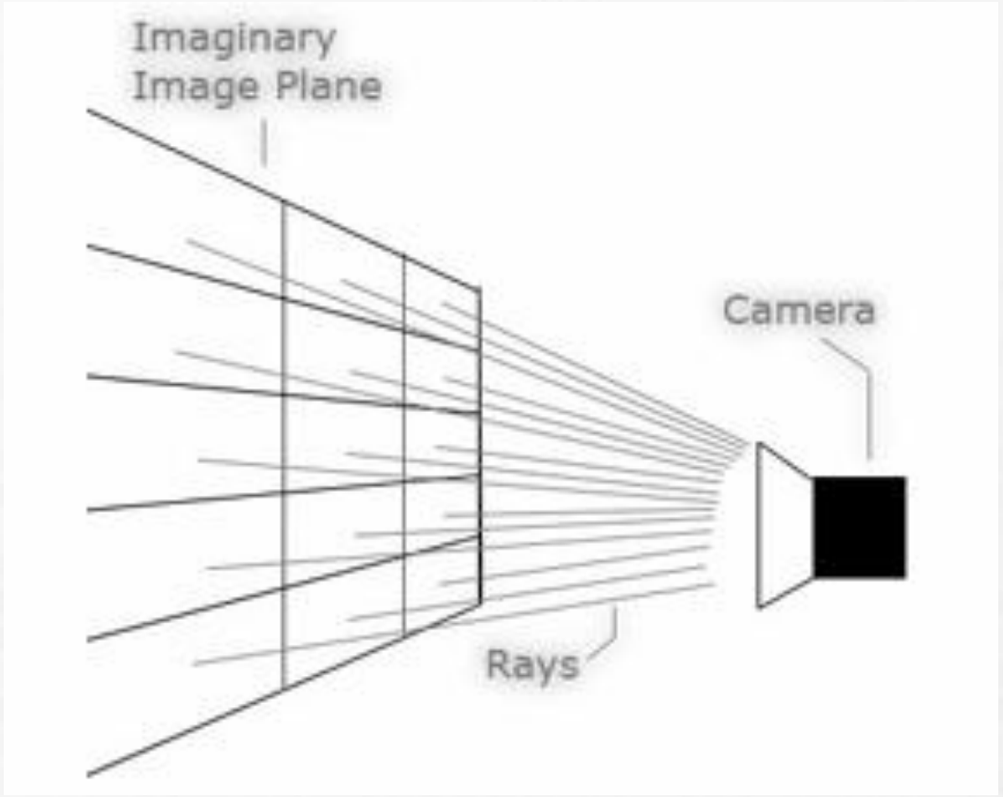
KỸ THUẬT DÒ TIA NGƯỢC (BACKWARD RAY TRACING)

NGUYÊN LÝ

Tia được bắn từ camera qua từng pixel trên một mặt phẳng hình ảnh ảo vào cảnh 3D.

ƯU ĐIỂM

Đảm bảo mọi tia bắn ra đều hướng về người xem, tiết kiệm tài nguyên mà không làm giảm chất lượng hình ảnh



KỸ THUẬT DÒ TIA NGƯỢC (BACKWARD RAY TRACING)

CÁC BƯỚC THỰC HIỆN

BƯỚC 1: TẠO TIA SÁNG TỪ CAMERA

Đối với mỗi pixel, tạo một tia sáng hướng về vị trí của pixel trên mặt phẳng ảnh tưởng tượng.

BƯỚC 2: KIỂM TRA GIAO CẮT

Đối với mỗi đối tượng trong cảnh, kiểm tra xem tia sáng có giao cắt với nó hay không.

BƯỚC 3: XÁC ĐỊNH ĐỐI TƯỢNG GẦN NHẤT

Nếu có giao cắt, chỉ ghi lại đối tượng nếu nó là đối tượng gần nhất bị giao cắt.

BƯỚC 4: TÔ BÓNG PIXEL

Sau khi tất cả các đối tượng đã được kiểm tra, hãy tô bóng pixel hình ảnh dựa trên điểm giao nhau với đối tượng gần nhất và thông tin của cảnh (ví dụ: chất liệu đối tượng, thông tin ánh sáng, v.v.).

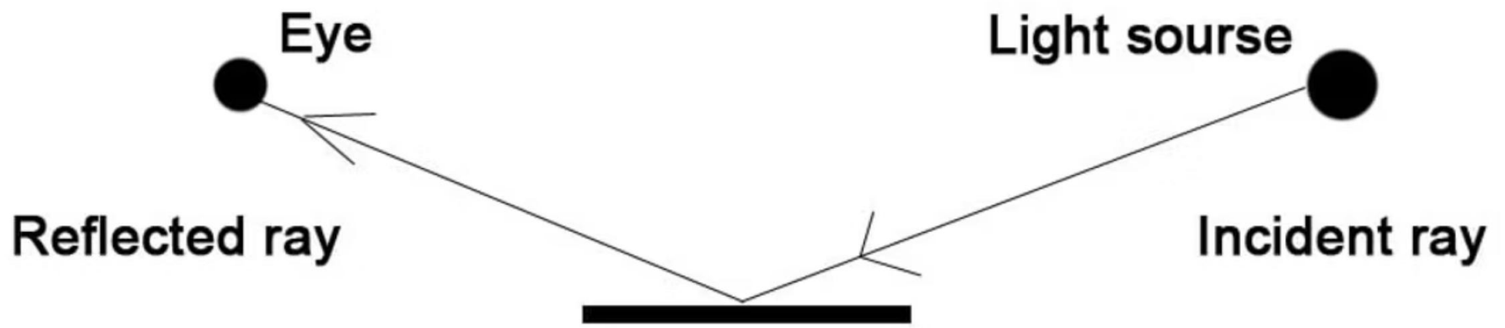
BƯỚC 5: LƯU KẾT QUẢ

Sau khi hoàn tất, hãy lưu kết quả hoặc sử dụng hình ảnh theo một cách có ý nghĩa khác.

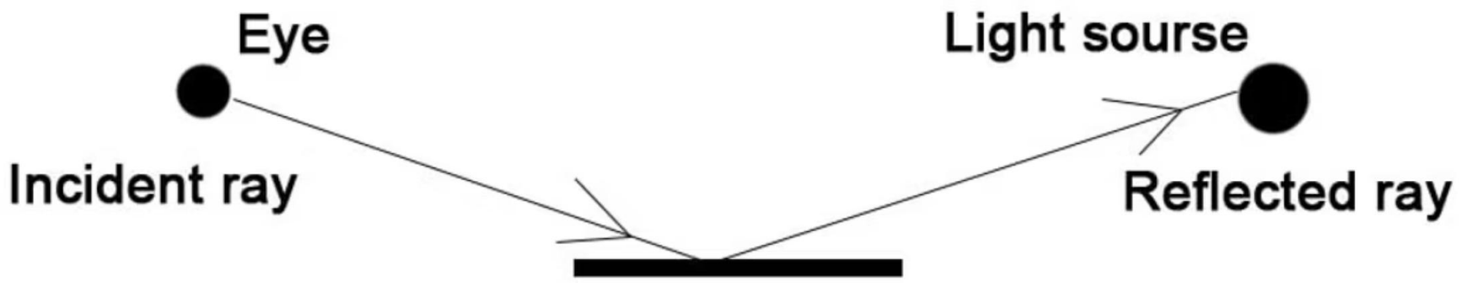
```
    function Backward_RayTracing(scene)
    {
       foreach(pixel in image)
       {
          ray = Calculate_Ray(pixel);
          closestObject = null;
          minDist = "Some Large Value";

          foreach(object in scene)
          {
             if(ray->Intersects(object) && object->distance <
minDist)
             {
                minDist = object->distance;
                point_of_intersect = ray->Get_Intersection();
                closestObject = object;
             }
          }

          if(closestObject != null)
          {
             foreach(light in scene)
             {
                Shade_Pixel(point_of_intersect, light,
closestObject);
             }
          }
       }

       Save_Render("output.jpg");
    }
```

KẾT LUẬN:

Đảm bảo rằng mọi tia được bắn ra đều hướng về phía người xem, giúp tiết kiệm đáng kể tài nguyên so với phương pháp tiến mà không làm giảm chất lượng hình ảnh.

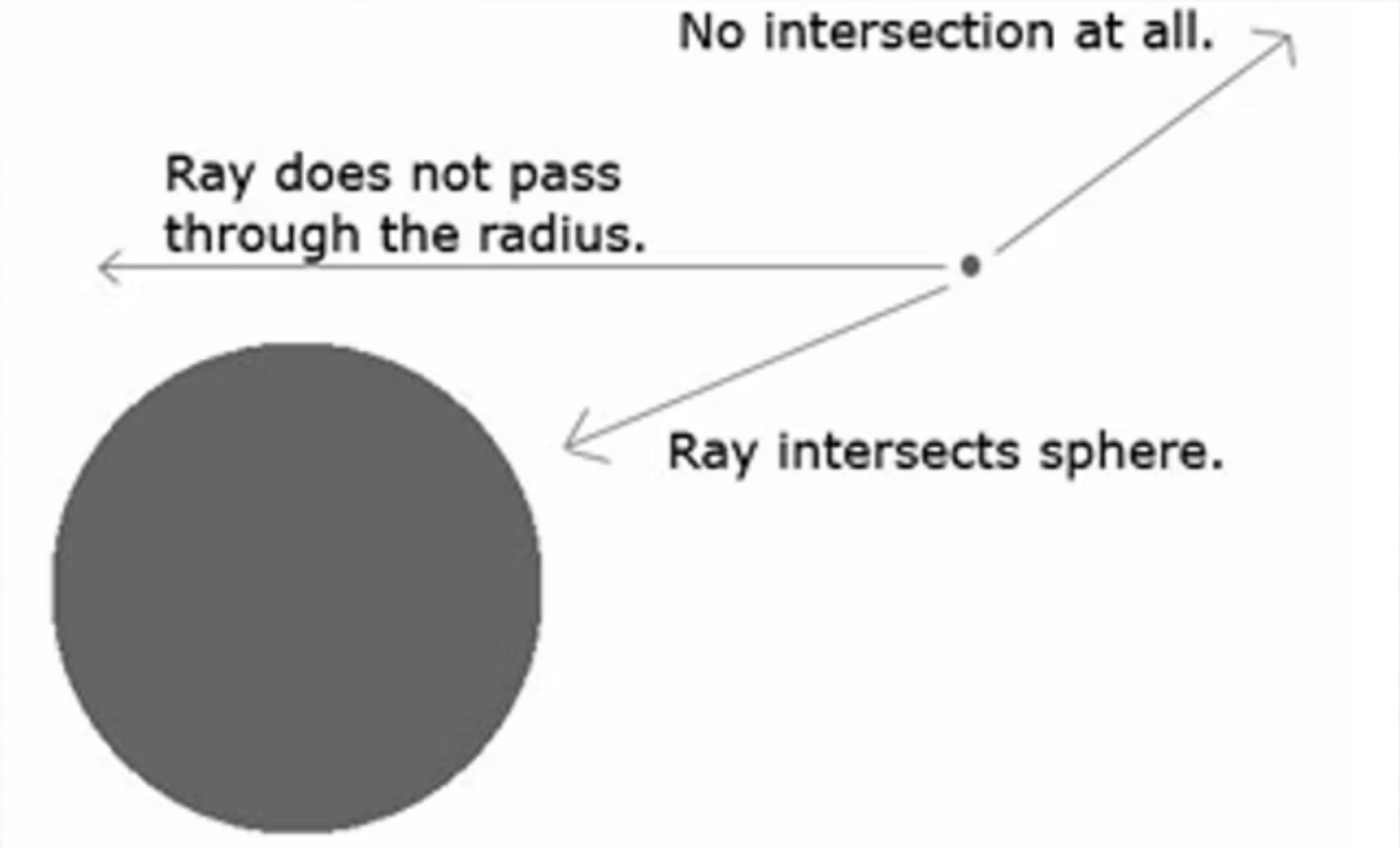


Forward Ray Tracing



Backward Ray Tracing

CÁC ĐỐI TƯỢNG CƠ BẢN - HÌNH CẦU (SPHERES)



ĐẶC ĐIỂM

Phổ biến vì tính toán rất nhanh, dễ biểu diễn toán học.

ỨNG DỤNG

Thường dùng làm vật thể đại diện để xác định nhanh va chạm trong game.

THUẬT TOÁN DÒ TIA HÌNH CẦU

1

TÍNH TOÁN VECTƠ:

Đầu tiên, xác định vectơ nằm giữa tâm của hình cầu và điểm gốc của tia sáng.

2

KIỂM TRA GÓC:

Tính toán **góc cosine** giữa vectơ tia-đến-tâm và hướng của tia. Nếu góc này nhỏ hơn 0, điều đó có nghĩa là tia sáng đang đi theo hướng ngược lại và không bao giờ chạm vào hình cầu.

3

KIỂM TRA BÁN KÍNH:

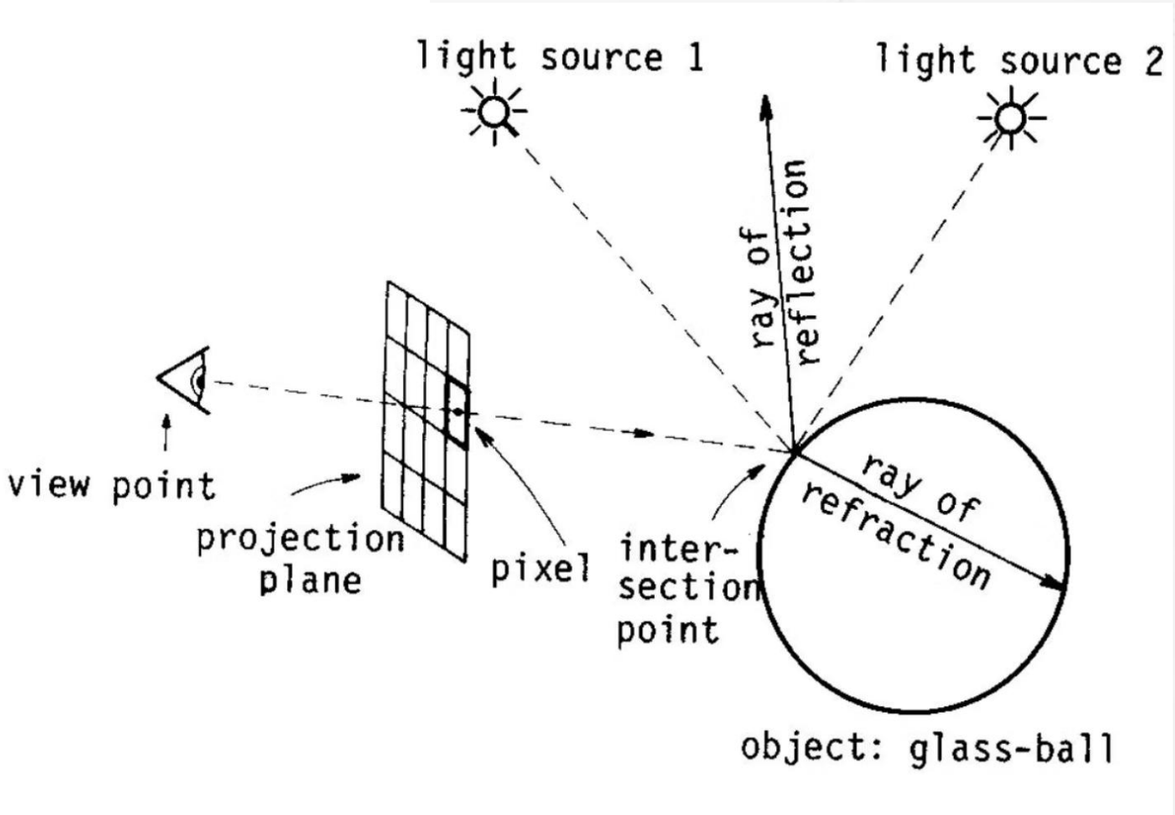
Nếu bài kiểm tra góc vượt qua, ta tiếp tục sử dụng bình phương bán kính của hình cầu, độ dài của vectơ tia-đến-tâm và bình phương của góc để xác định xem tia có xuyên qua khu vực bán kính của hình cầu hay không.

4

XÁC ĐỊNH KHOẢNG CÁCH:

Nếu có giao cắt, **khoảng cách giao điểm** sẽ được tính bằng góc giữa vectơ tia-đến-tâm và hướng tia, trừ đi căn bậc hai của kết quả kiểm tra bán kính

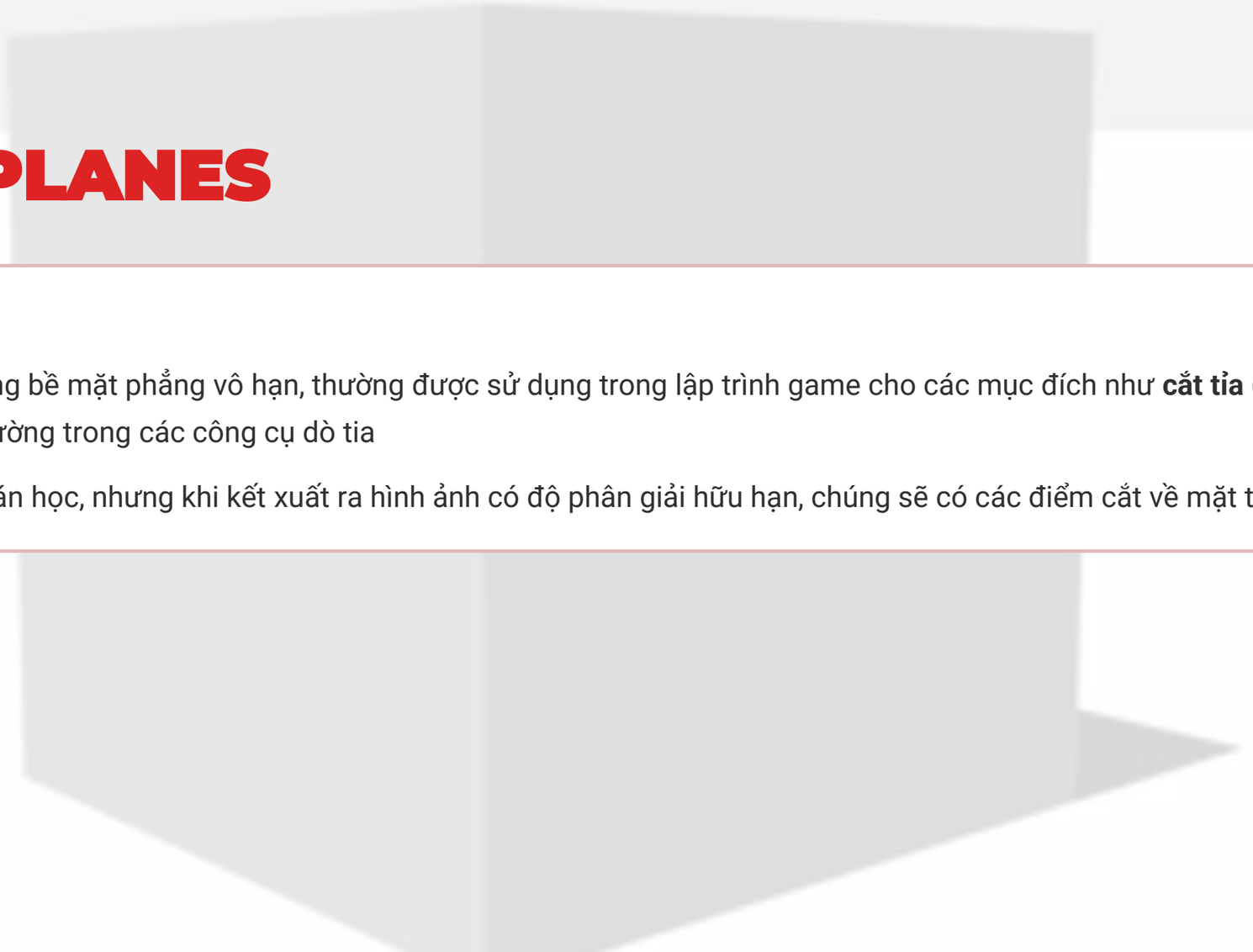
```
function Intersect(Sphere sphere, Ray ray, float *dist)
{
   Vector3D rsVec = sphere.center - ray.origin;

   float testA = rsVec.Dot3(ray.direction);

   if(testA < 0)
      return false;

   float rsLength = rsVec.Dot3(rsVec);

   float testB = (sphere.radius * sphere.radius) - rsLength +
                 (testA * testA);

   if(testB < 0)
      return false;

   if(dist != null)
      *dist = testA – square_root(testB);

   return true;
}
```



TRACING PLANES

Mặt phẳng (Planes) là những bề mặt phẳng vô hạn, thường được sử dụng trong lập trình game cho các mục đích như **cắt tỉa (culling)**, **phát hiện va chạm** hoặc làm sàn và tường trong các công cụ dò tia

Mặc dù là vô hạn về mặt toán học, nhưng khi kết xuất ra hình ảnh có độ phân giải hữu hạn, chúng sẽ có các điểm cắt về mặt thị giác



CÁC ĐỐI TƯỢNG CƠ BẢN - MẶT PHẲNG & HÌNH TAM GIÁC

01

QUY TRÌNH KIỂM TRA GIAO CẮT

Để xác định một tia sáng có cắt mặt phẳng hay không, tài liệu đưa ra các bước cụ thể:

1. Tính góc giữa pháp vectơ của mặt phẳng và hướng của tia.

2. Kiểm tra giá trị tuyệt đối của góc này; nếu nó nhỏ hơn 0 (hoặc cực kỳ nhỏ), tia đó đang song song với mặt phẳng và không có giao cắt,.

3. Nếu không song song, tính toán khoảng cách từ gốc tia đến mặt phẳng.

4. Chia khoảng cách từ gốc tia đến mặt phẳng cho góc cosine giữa hướng tia và pháp vectơ để xác định tia đang hướng về phía mặt phẳng hay hướng đi nơi khác.

5. Nếu tia không song song và đang hướng về phía mặt phẳng, nó chắc chắn sẽ giao cắt với mặt phẳng đó

```
function Intersect(Plane plane, Ray ray, float *dist)
{
   float rayPlaneAngle = plane.normal.Dot3(ray.direction);

   if(fabs(rayPlaneAngle) < 0.00001f)
      return false;

   float originD = -(plane.normal.Dot3(ray.origin) + plane.d);

   float intersectDist = originD / rayPlaneAngle;

   if(intersectDist < 0.0f)
      return false;

   if(dist != NULL)
      *dist = intersectDist;

   return true;
}
```

TRACING TRIANGLES

- Chi phí tính toán cao hơn hình cầu/hình hộp.
- Trong game, thường được bao quanh bởi các khối bao (bounding volumes) để tránh kiểm tra trực tiếp.
- Quy trình: Xác định cạnh → Tính tích có hướng → Kiểm tra hướng tia và mặt phẳng.

```
function Intersect(Triangle triangle, Ray &ray, float *dist)
{
   Vector3D vecAB = triangle.p2 - triangle.p1;
   Vector3D vecAC = triangle.p3 - triangle.p1;

   Vector3D origin = ray.origin;
   Vector3D direction = ray.direction;
   Vector3D cross = direction.CrossProduct(vecAC);

   float det = vecAB.Dot3(cross);

   if(det < 0.0001f)
      return false;

   Vector3D rayPointVec = origin - triangle.p1;

   float test1 = rayPointVec.Dot3(cross);

   if(test1 < 0.0f || test1 > det)
      return false;

   Vector3D cross2;
   cross2 = rayPointVec.CrossProduct(vecAB);
   float test2 = direction.Dot3(cross2);

   if(test2 < 0.0f || test1 + test2 > det)
      return false;

   float inverseDet = 1.0f / det;
    if(dist != NULL)
       {
          *dist = vecAC.Dot3(cross2);
          *dist *= inverseDet;
       }

    return true;
 }
```

Quy trình để xác định một tia sáng có cắt một hình tam giác hay không bao gồm các bước sau

- **Xác định các cạnh:** Lấy hai cạnh từ tam giác (ví dụ: vectơ từ điểm 1 đến điểm 2 và từ điểm 1 đến điểm 3).
- **Tính tích vectơ:** Tính tích có hướng (cross product) giữa cạnh đầu tiên và hướng của tia sáng.
- **Kiểm tra hướng tia:** Nếu góc cosine giữa kết quả tích vectơ vừa tính và cạnh đầu tiên nhỏ hơn 0, điều đó có nghĩa là tia sáng đi ra ngoài cạnh và bài kiểm tra thất bại.
- **Kiểm tra mặt phẳng:** Tiếp tục kiểm tra xem tia sáng có hướng ra xa mặt phẳng chứa tam giác hay không.
- **Kiểm tra bề mặt:** Cuối cùng, kiểm tra xem tia có nằm trong phạm vi các cạnh còn lại và thực sự đâm xuyên qua bề mặt tam giác hay không

MỘT SỐ ĐỐI TƯỢNG CƠ BẢN KHÁC

- Hình hộp và hình lập phương (Boxes and cubes)
- Hình vòng xuyến (Torus rings)
- Hình nón (Cones)
- Hình chóp (Pyramids)
- Hình đĩa (Disks)
- Hình trụ (Cylinders)

TRIỂN KHAI DÒ TIA THEO THỜI GIAN THỰC

Thách thức truyền thống: Ray Tracing từng bị coi là không khả thi cho thời gian thực do chi phí tính toán CPU cực cao và số lượng phép tính khổng lồ.

- **Sự chuyển mình:** Nhờ những tiến bộ trong kiến trúc máy tính và phần cứng, kỹ thuật này đang dần trở thành hiện thực và là lĩnh vực nghiên cứu đầy triển vọng cho các game thương mại.

- **So sánh:** Hiện nay, Ray Tracing vẫn chưa hoàn toàn thực tế để thay thế hoàn toàn Rasterization trong việc kết xuất các cảnh phức tạp ở độ phân giải cao, nhưng điều này đang thay đổi nhanh chóng

DÒ TIA TRÊN BỘ VI XỬ LÝ ĐA NHÂN (MULTI-CORE)

TÍNH SONG SONG LÝ TƯỞNG

Các tia được bắn vào cảnh một cách độc lập, không phụ thuộc lẫn nhau, giúp thuật toán dò tia cực kỳ phù hợp để chạy song song trên nhiều lõi xử lý.

CƠ CHẾ THỰC HIỆN

Các CPU hiện đại với nhiều lõi (kép, tứ, tám lõi) có khả năng xử lý đồng thời nhiều tia.

TỐI ƯU HÓA SIMD

Các tập lệnh SIMD (Single Instruction Multiple Data) đóng vai trò quan trọng trong việc xử lý song song hiệu quả các phép toán vector và ma trận, là nền tảng cho các tính toán trong Ray Tracing.

HỆ THỐNG MÁY CHỦ KẾT XUẤT(RENDERING FARMS)

MỞ RỘNG QUY MÔ

Thay vì một máy đơn lẻ, nhiều máy tính có thể kết nối qua mạng để chia sẻ công việc kết xuất, tối ưu hóa thời gian và hiệu quả.

ỨNG DỤNG THỰC TẾ

Kỹ thuật này thường được dùng trong điện ảnh để tạo ra các phim hoạt hình 3D phức tạp với độ chân thực cao và các hiệu ứng hình ảnh đặc biệt.

VÍ DỤ TIÊU BIỂU

Dự án Folding@Home kết nối người dùng PC và PS3 trên toàn thế giới để làm việc trên một mục tiêu tính toán chung là một ví dụ điển hình cho sức mạnh của tính toán phân tán.

DÒ TIA TRÊN BỘ XỬ LÝ ĐỒ HỌA (GPU)

SỬ DỤNG SHADER

Các card đồ họa hiện đại cho phép thực hiện dò tia hiệu quả thông qua các shader lập trình được như GLSL cho OpenGL, HLSL cho Direct3D

==> Mở ra khả năng tính toán đồ họa phức tạp ngay trên phần cứng.

TRUYỀN DỮ LIỆU QUA TEXTURE

Danh sách các vật thể trong cảnh có thể được truyền tải đến GPU dưới dạng các tệp ảnh (texture).

==> Giúp GPU truy cập dữ liệu hình học một cách nhanh chóng và tối ưu.

ĐỘ CHÍNH XÁC

Cần sử dụng các texture số thực (floating-point textures) để đảm bảo độ chính xác cho các tọa độ đỉnh của vật thể tránh sai lệch hình ảnh

HỆ THỐNG LỚN VÀ PHẦN CỨNG CHUYÊN DỤNG

1

RENDERING FARMS

Kết nối nhiều máy tính qua mạng để chia sẻ công việc (ứng dụng trong điện ảnh).

2

CHIP CHUYÊN BIỆT

Các đơn vị như SaarCor và RPU giúp đạt tốc độ đủ nhanh cho game thương mại.

3

KỸ THUẬT TỐI ƯU

- Phân vùng không gian (Spatial Partitioning).
- Giảm độ phân giải mẫu (Down-sampling).
- Giới hạn độ sâu đệ quy.
- Sử dụng thông tin tính toán trước (Precomputed Information)

KẾT LUẬN

Ray Tracing giúp chúng ta hiểu ánh sáng đúng theo bản chất vật lý của nó.

Cần hiểu rõ giới hạn phần cứng để áp dụng hiệu quả.

> **Ghi nhớ:** Rasterization vẫn tồn tại vì Ray Tracing quá chậm, không phải vì Ray Tracing sai.