Realismo Visual

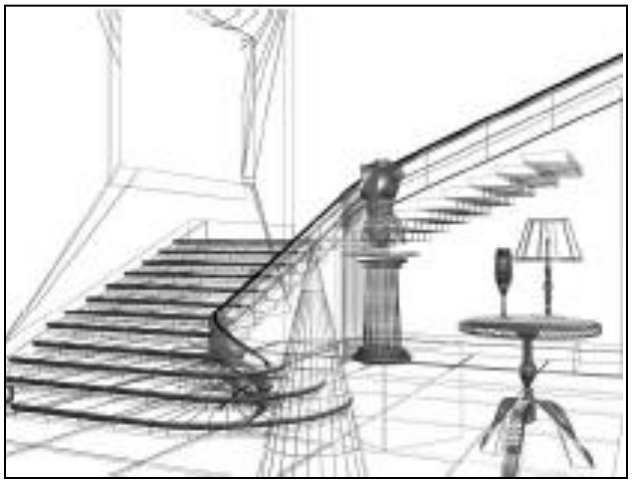
Luis Rivera

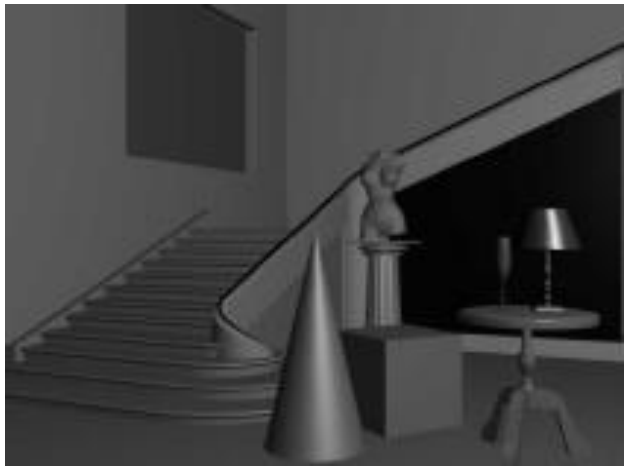
Realismos Visual

- Técnicas de tratamento computacional aplicadas a os objetos sintéticos
- Realismo fundamental em:
  - Simulações, entretenimento, educação, medicina, etc.
- Realismo em duas etapas:
  - Estática
    - Objetos e cenas estáticas com realismo fotográfico
  - Dinâmica
    - Cenas e objetos em movimento

Rendering

- No possui tradução para Português
- Def.
  - Criação sintética das cenas com realismo fotográfico, em termos da definição dos dados dos objetos que a compõem.
- Considerando a Geometria da cena, materiais, iluminação, etc.



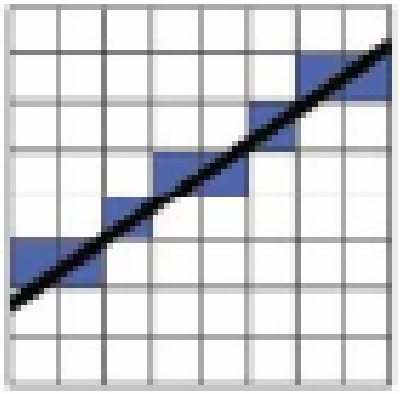


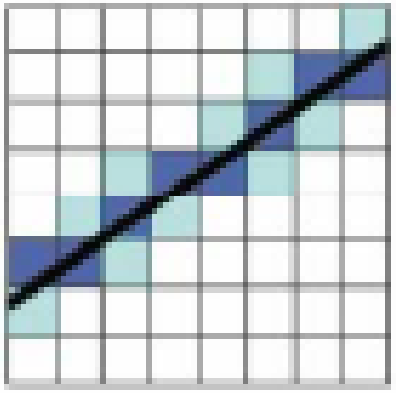
Fases de Processo de Realismo Visual

- Não todas as fases são usadas em todas as aplicações:
  - Construção do modelo
    - Modelado (informações necessárias para realismo visual)
  - Aparência tridimensional
    - Transformações (projeções e perspectivas, etc)
  - Eliminação de polígonos ou faces escondidas
    - Considerando posição relativa da cena e o observador
  - Recortes
    - Clipping (segmento da cena visível)
  - Rasterização
    - Objeto 3D transformado em pixel
  - Colorir cada pixel individualmente
    - Efeito de sombras, luz, brilho, transparência, textura, etc.

Rasterização

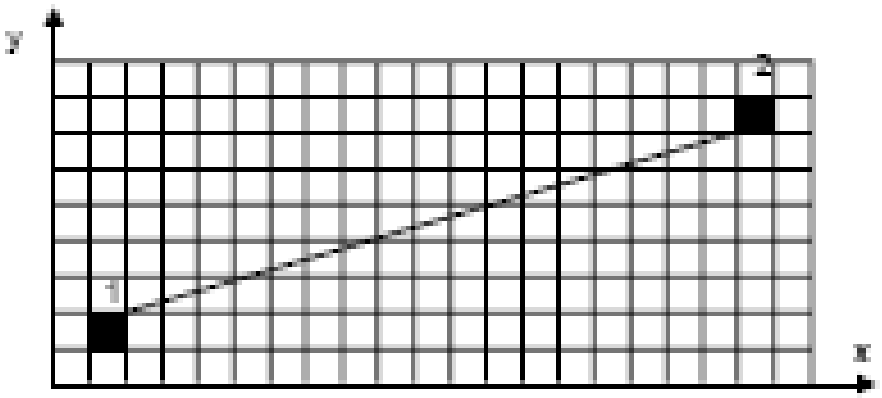
- Conversão da representação vetorial para matricial
- Podemos obter uma linha com uma aparência endentada (aliasing)
- Algoritmos de anti-aliasing
  - Algoritmo de Bresenham para linhas
  - Rasterização de polígonos
  - Preenchimento de polígonos por scanline
  - Remoção de linhas e superfícies escondidas



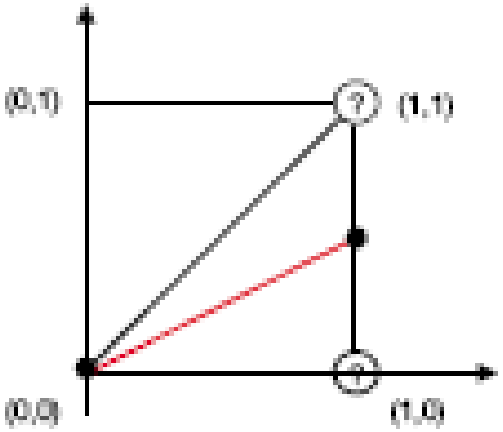


Rastering de retas

- Objetos definidos por pontos, retas, círculos, etc.
- Retas não são perfeitas
  - Aliasing
- Algoritmo de Bresenham
  - Linha entre 1=(x_1, y_1) y 2=(x_2,y_2)
  - Para cada pixel próximo à reta

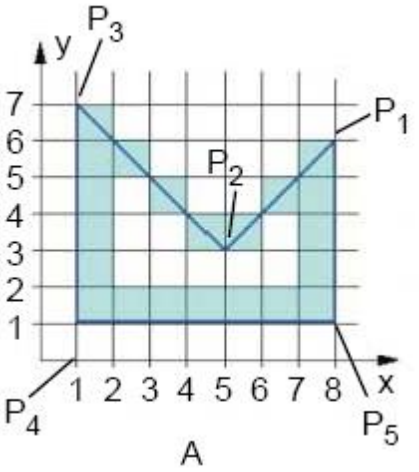



```
x= x1;  y= y1;
dx= x2 – x1;  dy= y2 – y1;
m= dy/dx;   e = m – 0.5;
FOR i=1… dx
        disenhaPonto(x,y)
         WHILE (e >= 0)
                    y = y + 1
                    e = e – 1
        x = x + 1
        e = e + m
```

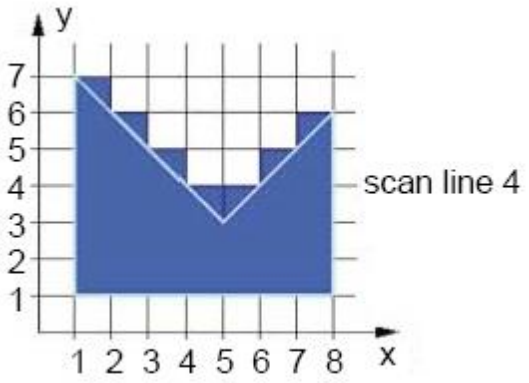


Rastering de Polígonos por Scan Line

- Objetos 3D são projetados num plano
- Determinados segmentos de arestas
  - Interseções
  - Processo de raster por segmento
  - Fig A
- Para preenchimento por scan line
  - Cada linha de scan line
    - Segmentada por arestas
    - Considerar qual é interno e qual externo
  - Fig B



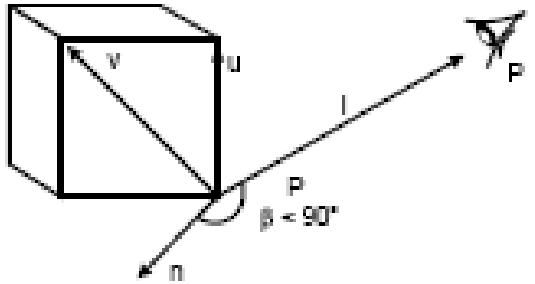
A

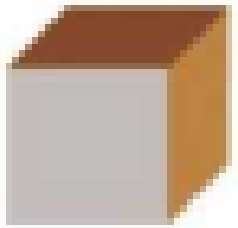


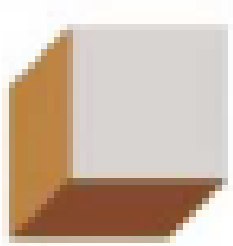
B

Remoção de linhas e superfícies escondidas

- Elementos dos objetos visíveis dependem da referencia do observador
- Objetos aproximados a faces planas (polígonos)

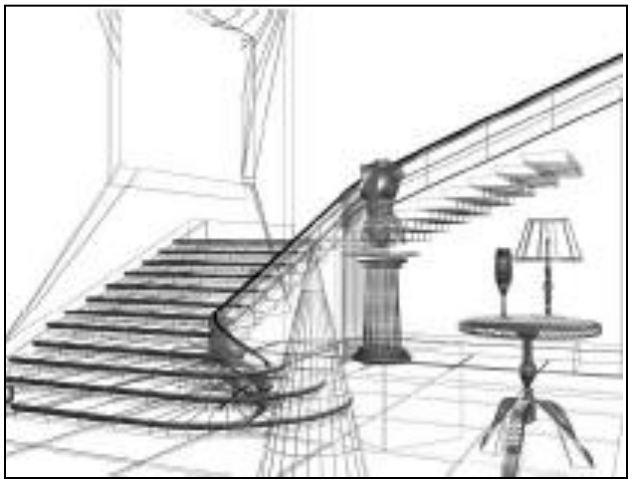


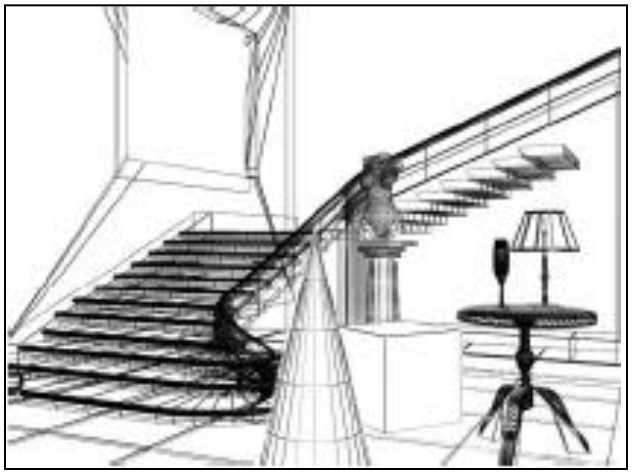




Remoção de linhas e superfícies escondidas

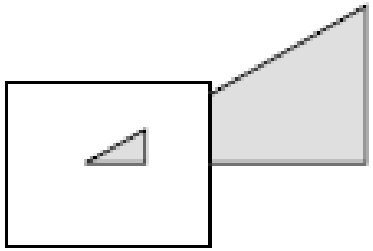
- Formas de representação das wire-frames não-visíveis:
  - Tracejadas com a mesma cor
  - Tracejadas com outra cor
  - Não tracejadas
    - Maiormente usada
- Algoritmos populares
  - Algoritmo do Pintor
  - Algoritmo de Eliminação de faces Ocultas pelo Cálculo Normal
  - Algoritmo Z-Buffer

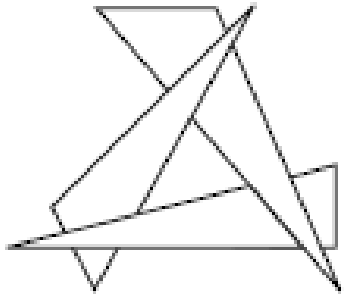




Remoção de linhas e superfícies escondidas

- Algoritmo do Pintor
  - Simula a forma como o pintor faria
    - Pintar objetos mais distantes do observador
  - Se face A bloqueia a visão de B, então B está mais distante que A.
- Algoritmo
  - Calcular a distancia ao observador de todas as faces poligonales da cena
  - Ordenar todos os polígonos pelo valor de sua distancia ao observador
  - Resolver as redundantes
  - Desenhar primeiro os polígonos que estão mais distantes do observador
- Dificuldades
  - Objetos parte visíveis





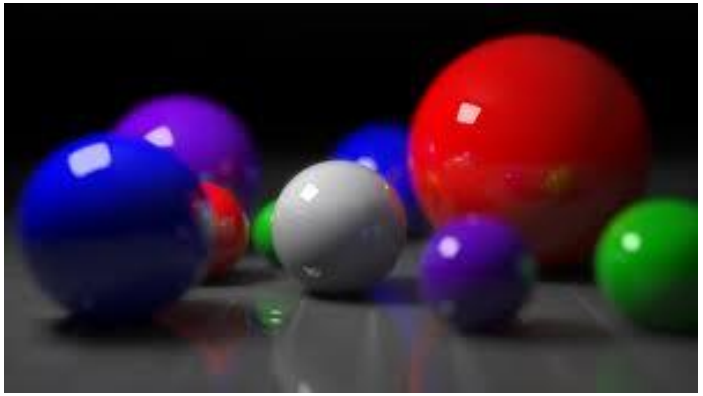
Remoção de linhas e superfícies escondidas

- Algoritmo de Eliminação de Faces Ocultas pelo Cálculo Normal
  - ângulo da normal com a superfície na direção do observador
- Teste de Visibilidade
  - Dois vetores de orientação (u e v) associados a cada face ou superfície
  - O vetor normal (n) de cada uma dessas faces ou superfícies
  - Vetor da linha de visibilidade (l)
  - Cálculo do ângulo (b) – vetor normal (n) e vetor da linha de visibilidade (l)
    - $n.l = |n| . |l| . \cos b$

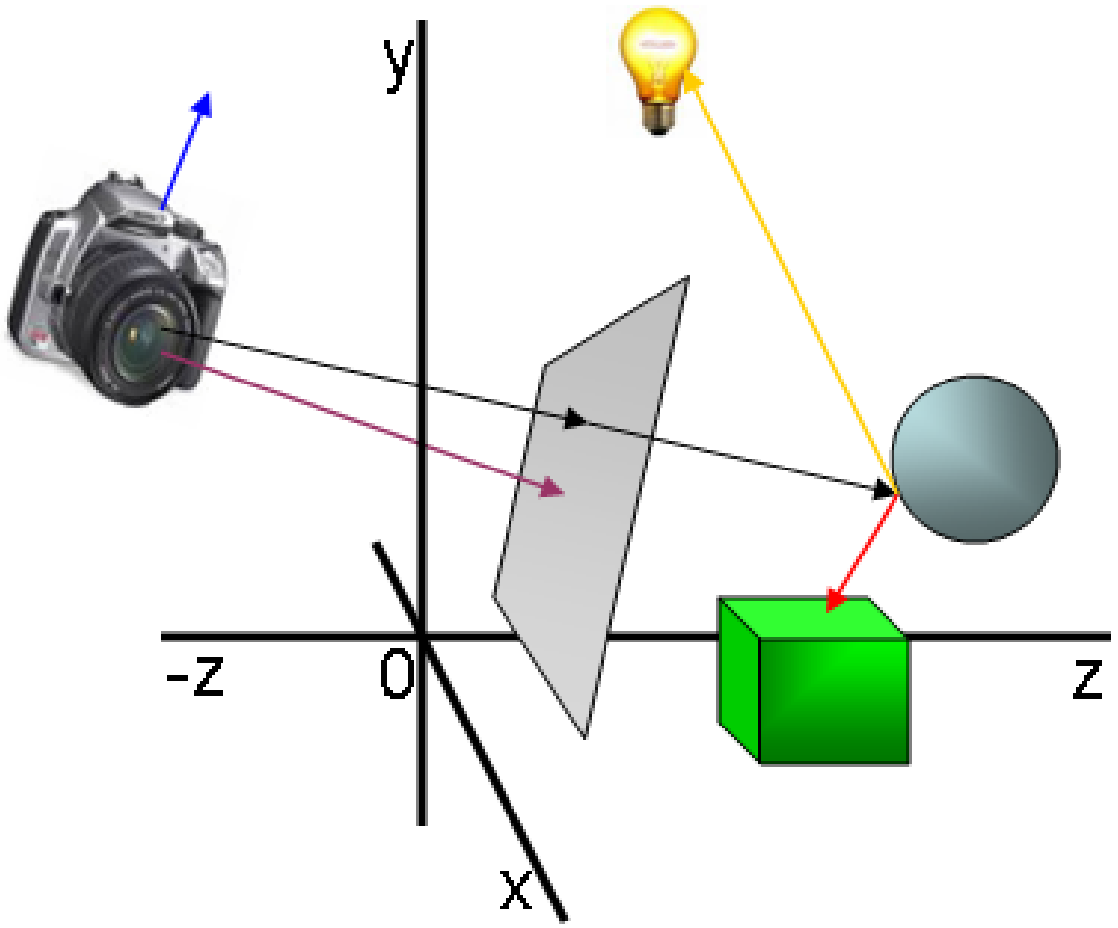
Remoção de linhas e superfícies escondidas

- Algoritmo de Eliminação de Faces Ocultas por Cálculo Normal
  - Ler as coordenadas dos objetos, considerando um ponto de referencia e armazenar em forma de matriz
  - Localizar no espaço da posição do observador
  - Calcular o vetor normal de cada face do objeto
  - Calcular o vetor da linha de visibilidade para cada face do objeto
  - Realizar a verificação de visibilidade
    - Si $n.l > 0$, a face estará visível
    - Si $n.l < 0$, a face estará invisível
  - Definir os cantos das faces do objeto e armazenar-los de formal matricial
  - Verificar os cantos visíveis, com seus respectivos posicionamentos
  - Tracejar as arestas das faces visíveis.

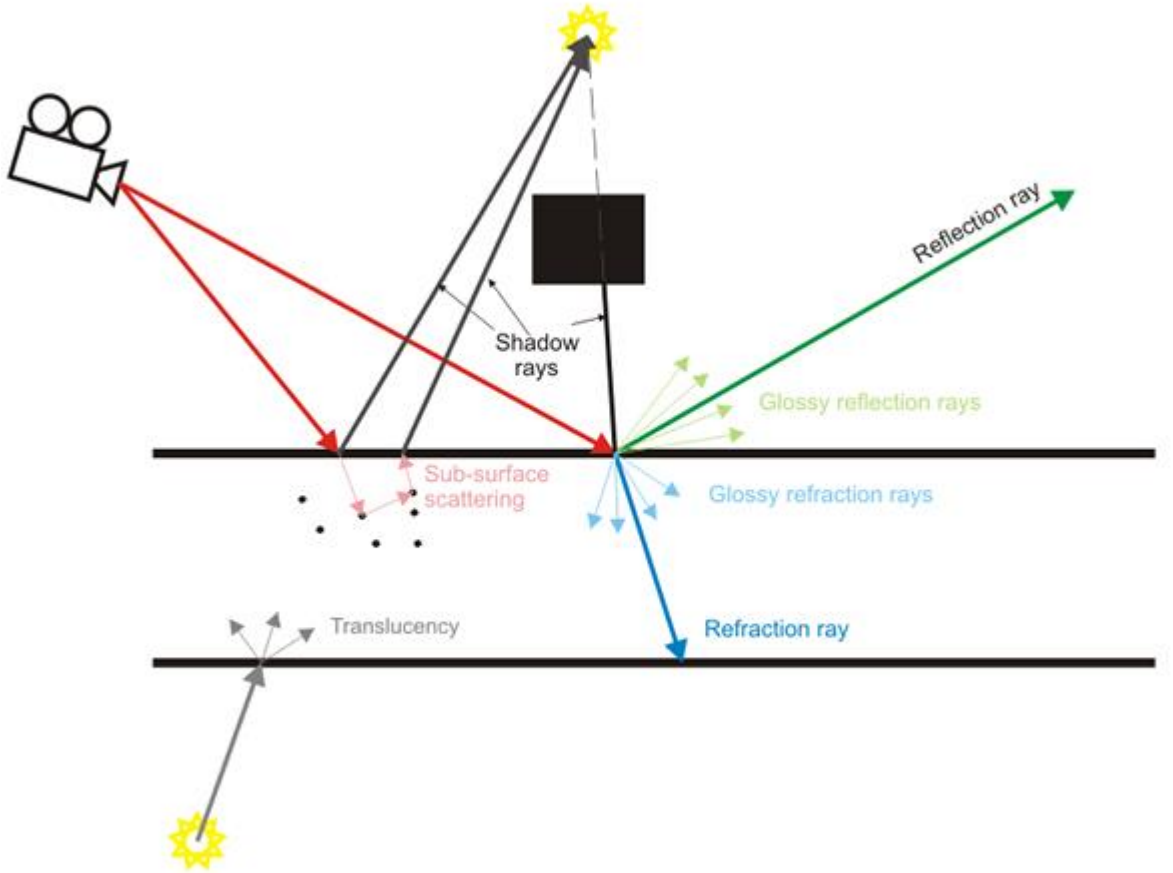
RayTracing



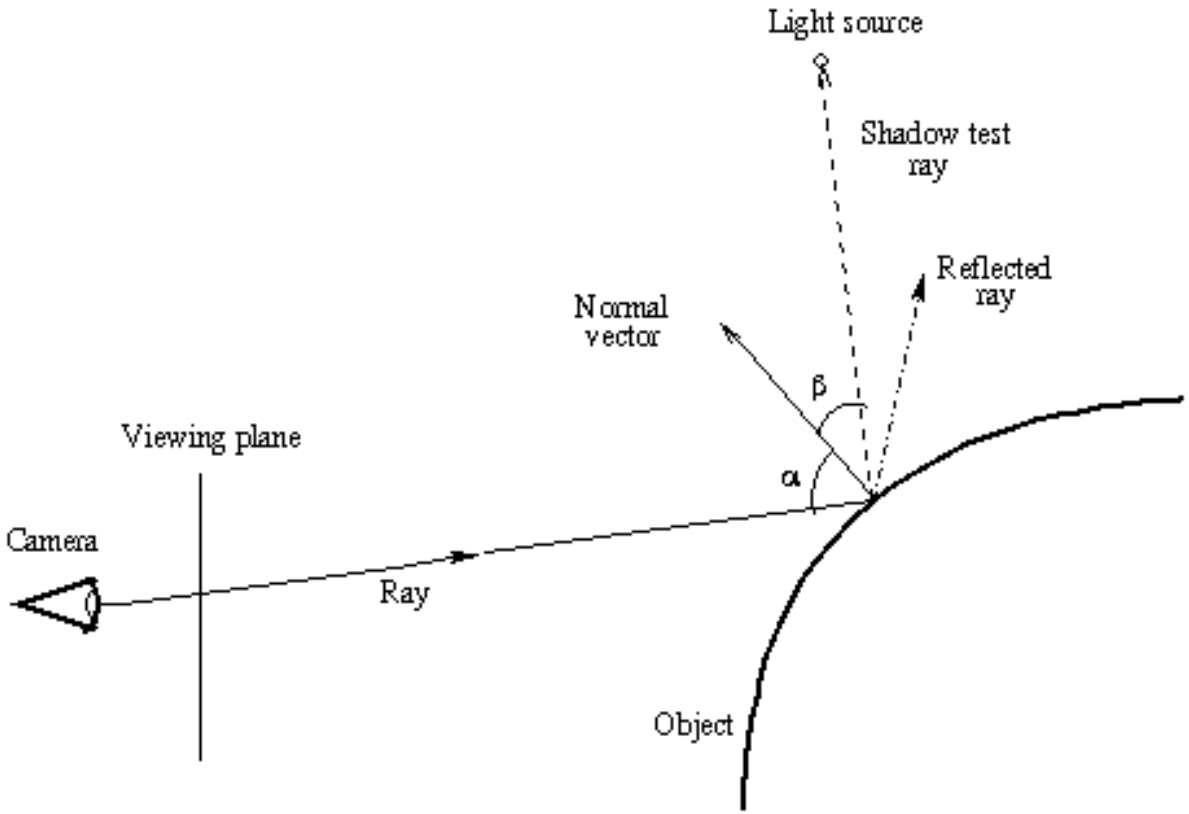
Ray Tracing



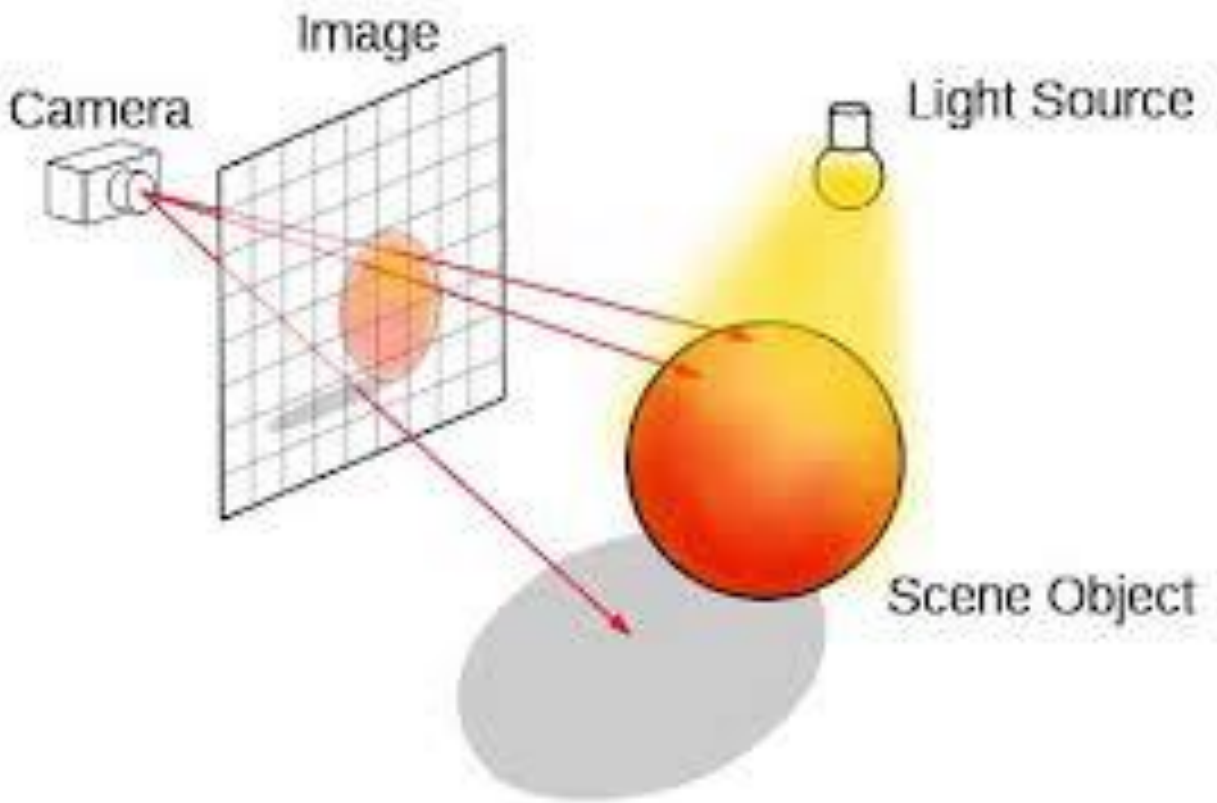
RayTracing


Reflection ray
Shadow rays
Glossy reflection rays
Sub-surface scattering
Glossy refraction rays
Translucency
Refraction ray

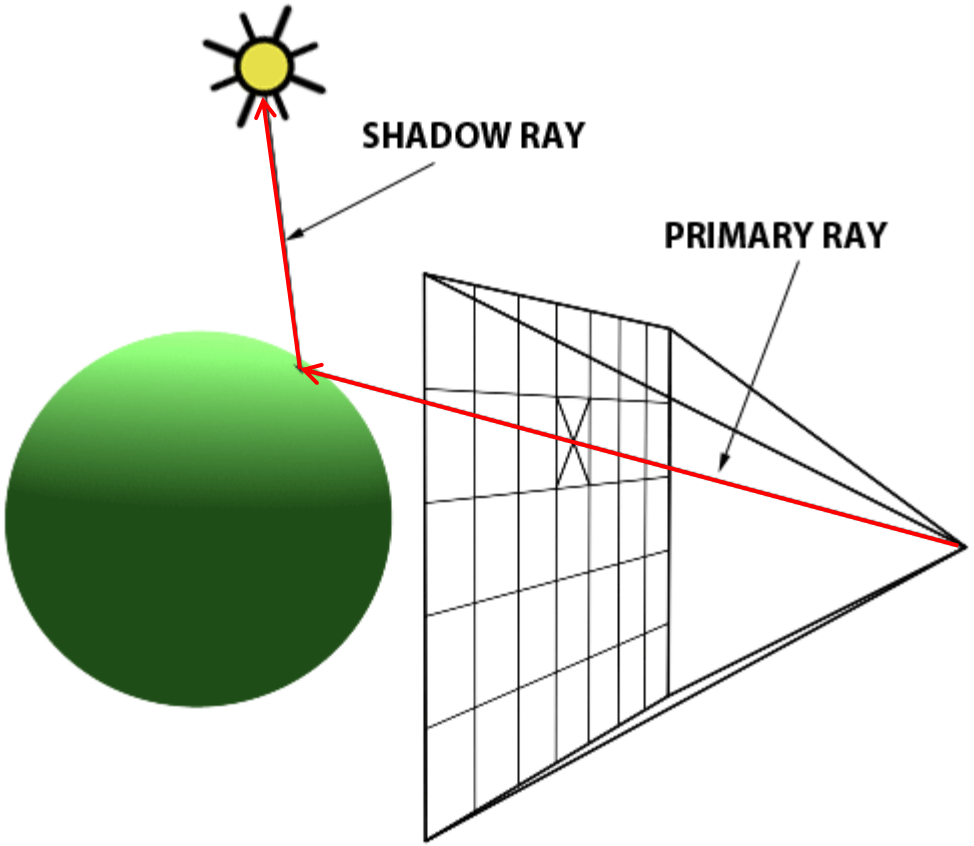
Ray Tracing


Light source
Shadow test ray
Reflected ray
Normal vector
β
α
Viewing plane
Camera
Ray
Object

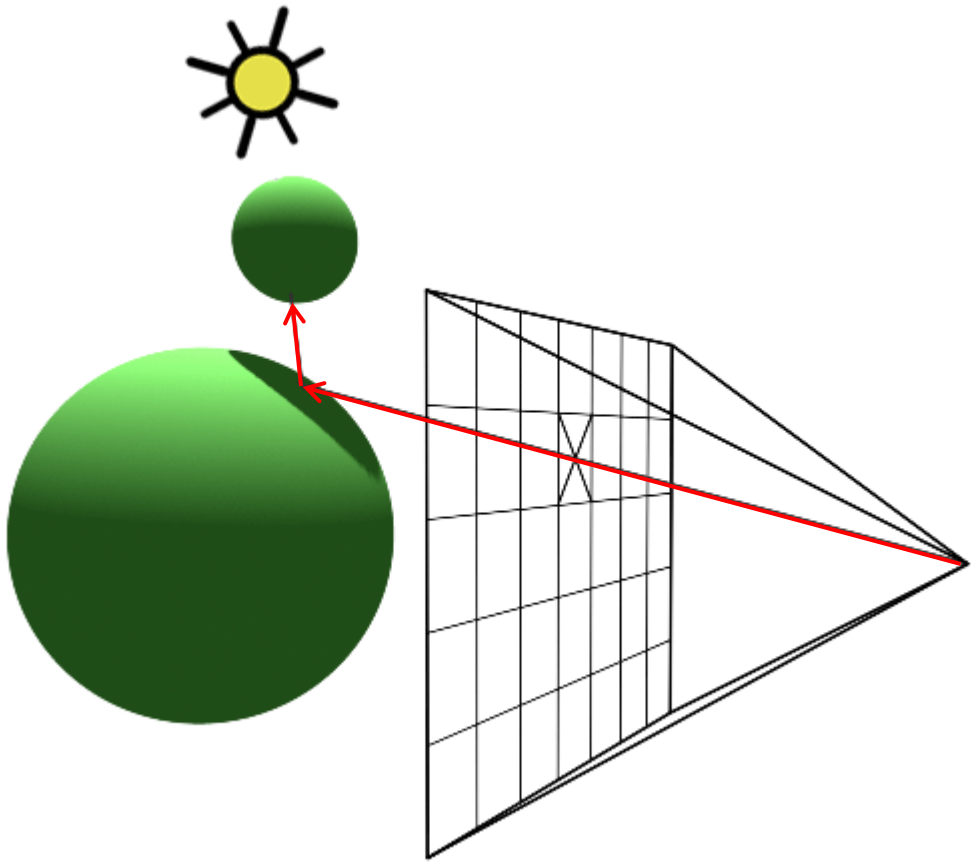
Ray Tracing



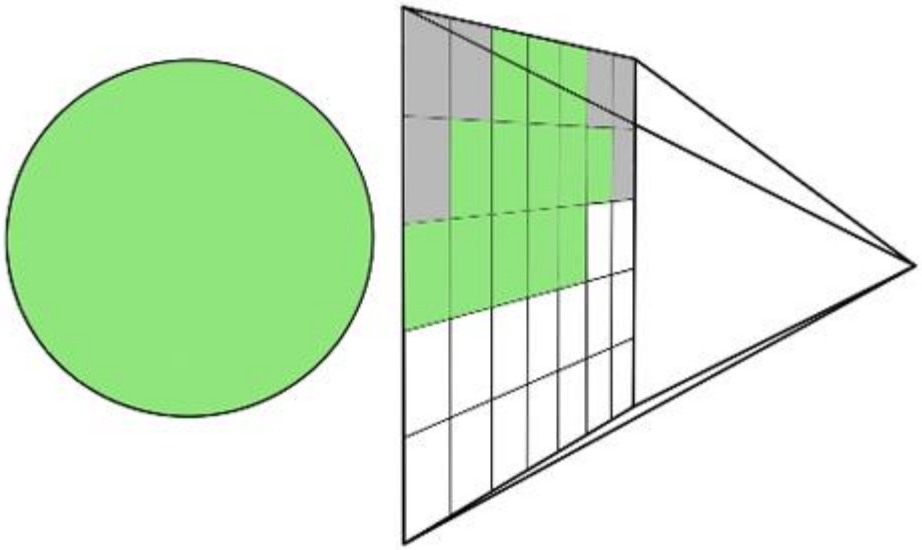
Interseção dos raios



Interseção dos raios

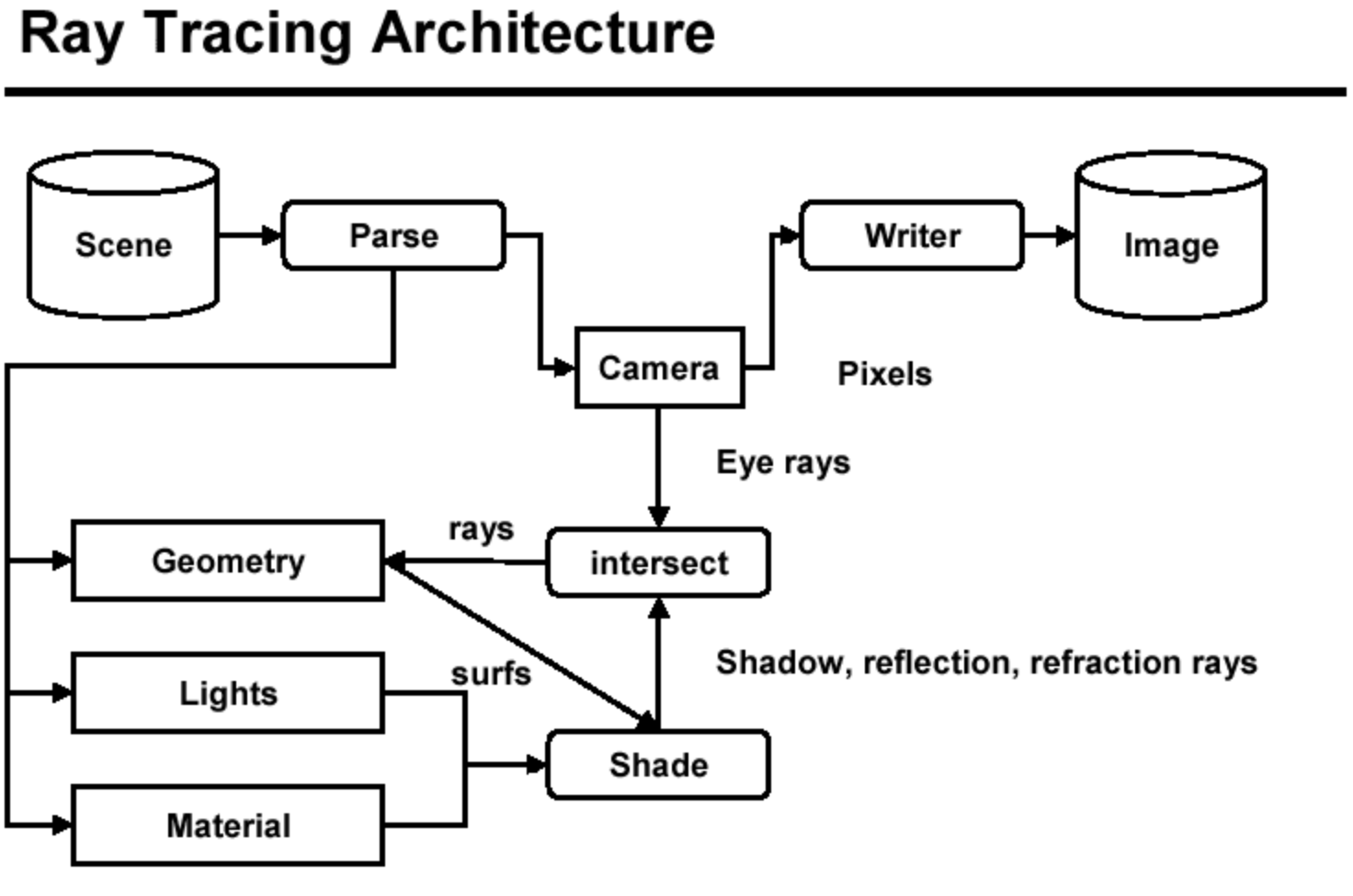


Rendering de um frame



```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
            // compute primary ray direction
        Ray primRay;
        computePrimRay(i, j, &primRay);
            // shoot prim ray in the scene and search for intersection
        Point pHit;
        Normal nHit;
        float minDist = INFINITY;
        Object object = NULL;
        for (int k = 0; k < objects.size(); ++k) {
            if (Intersect(objects[k], primRay, &pHit, &nHit)) {
                float distance = Distance(eyePosition, pHit);
                if (distance < minDistance) {
                    object = objects[k];
                    minDistance = distance; // update min distance
                }
            }
        }
        if (object != NULL) {
                // compute illumination
            Ray shadowRay;
            shadowRay.direction = lightPosition - pHit;
            bool isShadow = false;
            for (int k = 0; k < objects.size(); ++k) {
                if (Intersect(objects[k], shadowRay)) {
                    isInShadow = true;
                    break;
                }
            }
        }
        if (!isInShadow)
            pixels[i][j] = object->color * light.brightness;
        else
            pixels[i][j] = 0;
    }
}
```

RayTracing


Ray Tracing Architecture
Scene
Parse
Writer
Image
Camera
Pixels
Eye rays
rays
Geometry
intersect
Lights
surfs
Shadow, reflection, refraction rays
Shade
Material