|  |  |
| --- | --- |
|  | Fundamentos da Computação Gráfica |
|  |  |
| 16/06/2014 | Rendering |
|  | Aluno: Leonardo Henrique Camello do Nascimento  Professor: Marcelo Gattass |

Fundamentos da Computação Gráfica

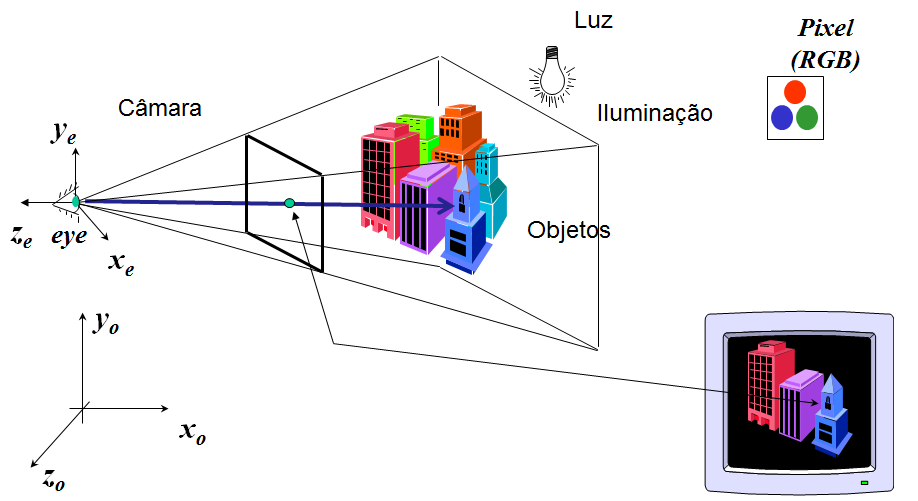
Rendering

# Introdução

O objetivo deste trabalho é implementar o algoritmo de traçado de raios e produzir algumas cenas demostrando os detalhes da implementação como luz ambiente, reflexão difusa e especular, e sombra.

# Traçado de raios

O algoritmo de ray tracing (traçado de raios) é utilizado para a síntese de imagens tridimensionais. Nele, temos um modelo simplificado de uma câmera que emite raios na direção da cena. Para cada pixel da tela é emitido um raio e através da interação dele com as luzes e os objetos da cena é retornada a cor do pixel, formando a imagem tridimensional.



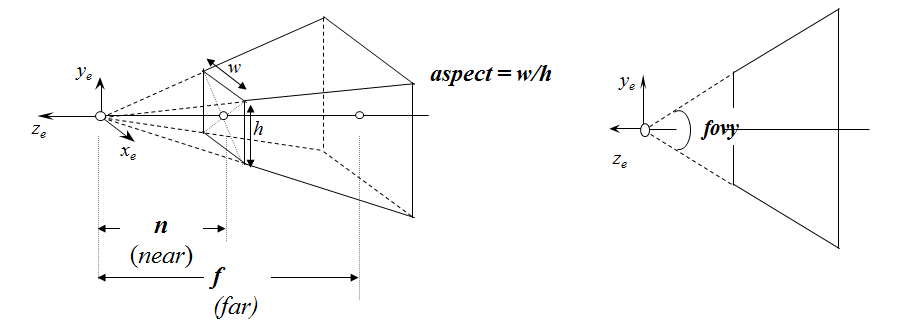
## Modelo da Câmera

No nosso modelo de câmera definimos alguns parâmetros internos primários: a distância dos planos *near* e *far*, a largura e a altura da tela, em pixels, e o campo de visão da câmera. Com eles podemos calcular outros parâmetros internos: a distância focal, e a largura e a altura da imagem em unidades do mundo.

Fundamentos da Computação Gráfica

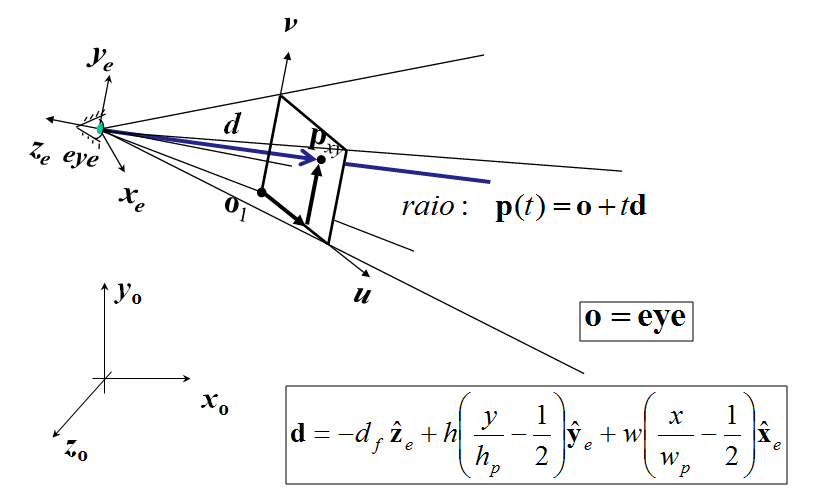
Rendering

Também precisamos definir alguns parâmetros externos da câmera, relativos ao posicionamento da mesma: os vetores *eye*, *center* e *up*. Com esses podemos calcular o sistema de coordenadas da câmera: os vetores *xe*, *ye* e *ze.*



## Lançamento de Raios

Os raios parametrizados (*p(t) = o + td*) tem a origem (*o*) como sendo a posição da camêra (*eye)* e a direção (*d*) é dada em função do pixel (*x, y*) da tela e dos vetores que definem o observador (*xe*, *ye* e *ze*):



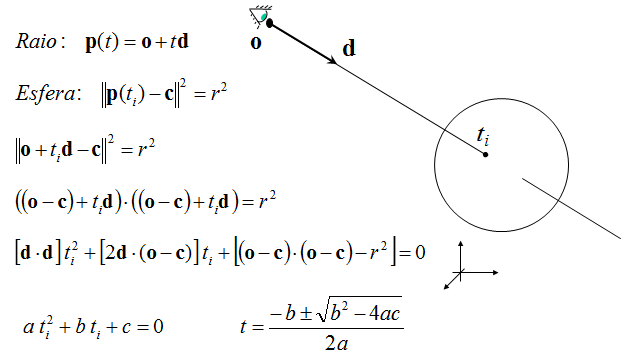
Fundamentos da Computação Gráfica

Rendering

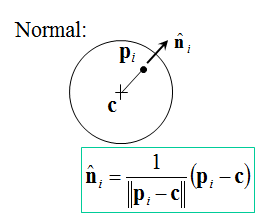
## Modelagem dos Objetos

Foram utilizados dois tipos de objetos parametrizados no trabalho: esferas e caixas alinhadas com os eixos.

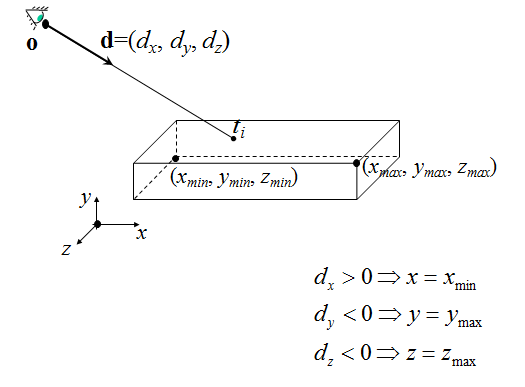
Sabendo que os raios parametrizados são da forma *p(t) = o + td*, podemos encontrar a interseção do raio com a esfera, na forma , como mostra a figura:



E a normal da esfera é dada por:



Para as caixas alinhadas com os eixos, podemos encontrar a interseção do raio com a caixa:



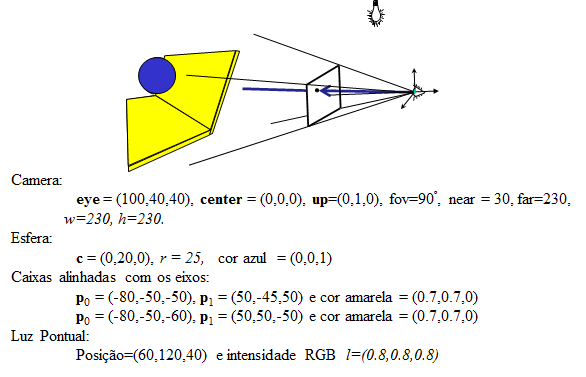
Fundamentos da Computação Gráfica

Rendering

E a normal é relativa à face em que o raio intersecta a caixa. Como a caixa é alinhada com os eixos, as normais podem assumir os seguintes valores:

## Uma Cena Simples

Para demonstrar os detalhes do algoritmo escolhemos uma cena simples:



Nessa cena temos a definição das entradas necessárias para configurarmos os parâmetros da câmera, o posicionamento e a intensidade da luz, o posicionamento, o raio e a cor difusa da esfera, e os pontos mínimos e máximos, e as cores das caixas alinhadas com os eixos.

## Luz Ambiente

No cálculo da contribuição da luz ambiente, para a cor do pixel, temos:

Fundamentos da Computação Gráfica

Rendering

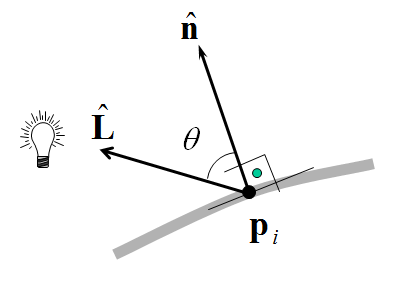
Onde são os valores da intensidade da luz ambiente e são os valores da cor difusa do objeto intersectado.

## Componente de Reflexão Difusa

No cálculo da contribuição da componente de reflexão especular, para a cor do pixel, temos:

Onde são os valores da intensidade da luz em questão. Pode haver mais de uma luz na cena. E são os valores da cor difusa do objeto intersectado.

O vetor é a normal do objeto no ponto de interseção e é o vetor cuja origem é o ponto de interseção e a sua direção é a posição da luz em questão.



## Componente de Reflexão Especular

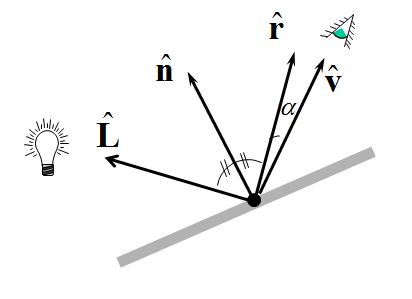
No cálculo da contribuição componente de reflexão especular, para a cor do pixel, temos:

Onde são os valores da intensidade da luz em questão. Pode haver mais de uma luz na cena. E são os valores da cor especular do objeto intersectado.

Fundamentos da Computação Gráfica

Rendering

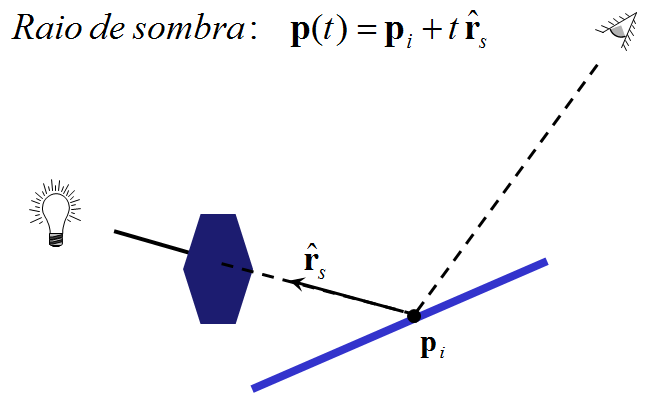
O vetor é o vetor refletido sobre a normal e é o vetor cuja origem é o ponto de interseção e a sua direção é a posição da câmera.



A reflexão do vetor sobre a normal é dada por:

## Sombra

Para descobrirmos se um pixel está na sombra, lançamos um raio cuja origem é o ponto de interseção com o objeto da cena e a direção é a posição da luz. Caso esse raio intersecte algum objeto utilizamos para a cor desse pixel, apenas a contribuição da luz ambiente.

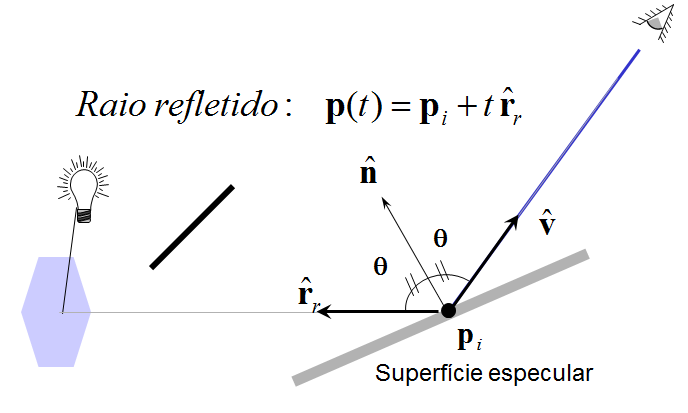


Fundamentos da Computação Gráfica

Rendering

## Reflexão

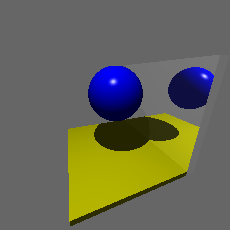
Se o objeto é um objeto refletor, calculamos essa reflexão lançando um raio cuja origem é o ponto de interseção com o objeto da cena e a direção é a posição da câmera refletida sobre a normal.



## Modelo de Várias Luzes, Sombra e Superfícies Refletoras

Para o modelo de várias luzes, considerando a sombra e superfícies refletoras, a cor do pixel será a soma da contribuição da luz ambiente, mais o somatório, para cada luz, da soma das contribuições difusas e especulares, caso a luz chegue na superfície, mais a componente da superfície refletora, que é a cor do objeto refletido multiplicado por um fator de redução.

Abaixo podemos ver o resultado esperado para a nossa cena simples:



Fundamentos da Computação Gráfica

Rendering

# A Aplicação

## Overview

O programa escrito em Javascript, inicialmente, faz toda a configuração da câmera e da cena. Com a câmera, as luzes e os objetos da cena posicionados, o Javascript gera e compila o Vertex Shader e o Fragment Shader em GLSL. O algoritmo do Ray Tracing foi implementado nos Shaders para uma melhora no desempenho da aplicação.

Foi criada uma página em HTML onde o usuário consegue escolher e visualizar cenas predefinidas, e também mudar a posição da câmera. O desenho das imagens da cena foram feitas com o WebGL.

## O Modelo da Câmera e a Cena

O modelo da câmera que foi utilizado foi o mesmo descrito acima. O construtor da classe câmera ficou da seguinte maneira:

function Camera(fov, near, far, wp, hp, eye, center, up) {

// Parâmetros internos primários

this.fov = fov;

this.near = near; this.far = far;

this.wp = wp; this.hp = hp;

// Parâmetros internos derivados

this.df = this.near;

this.h = 2 \* this.df \* Math.tan(this.fov / 2.0 \* Math.PI / 180);

this.w = (this.wp / this.hp) \* this.h;

// Parâmetros externos

this.eye = eye;

this.center = center;

this.up = up;

// Sistema da câmera

this.ze = new THREE.Vector3();

this.ze.subVectors(this.eye, this.center);

this.ze.normalize();

this.xe = new THREE.Vector3();

this.xe.crossVectors(this.up, this.ze);

this.xe.normalize();

this.ye = new THREE.Vector3();

this.ye.crossVectors(this.ze, this.xe);

this.ye.normalize();

}

Fundamentos da Computação Gráfica

Rendering

A cena consiste em um conjunto de luzes e objetos como pode ser visto:

function Scene() {

this.lights = [];

this.objects = [];

}

Dentre esse objetos podemos ter esferas e caixas:

function Sphere(id, radius, center) {

SceneObject.apply(this, arguments);

this.radius = radius;

this.center = center;

this.name = 'Sphere' + id;

this.setDiffuseColor(1.0, 1.0, 1.0);

this.setSpecularColor(1.0, 1.0, 1.0);

this.radiusStr = 'u\_Radius' + this.name;

this.centerStr = 'u\_Center' + this.name;

this.intersectStr = 'Intersect' + this.name;

}

function Box(id, lower, upper) {

SceneObject.apply(this, arguments);

this.lower = lower;

this.upper = upper;

this.name = 'Box' + id;

this.setDiffuseColor(1.0, 1.0, 1.0);

this.setSpecularColor(1.0, 1.0, 1.0);

this.lowerStr = 'u\_Lower' + this.name;

this.upperStr = 'u\_Upper' + this.name;

this.intersectStr = 'Intersect' + this.name;

}

O modelo da luz também é bem simples:

function Light(id, position) {

SceneObject.apply(this, arguments);

this.position = position;

this.name = 'Light' + id;

this.setColor(1.0, 1.0, 1.0);

this.positionStr = 'u\_Position' + this.name;

}

Fundamentos da Computação Gráfica

Rendering

## O Algoritmo

Com a cena configurada, é passado ao vertex shader os 4 vértices dos cantos da tela. Assim, cada pixel da tela se torna um fragmento e será computado pelo fragment shader.

É no fragment shader que ocorrem os cálculos do algoritmo. Ele recebe a posição da câmera e a direção do raio para aquele pixel.

'{                                                  \n' +

'void main()                                        \n' +

'   gl\_FragColor = vec4(trace(u\_Eye, v\_Ray), 1.0); \n' +

'}                                    \n';

O método trace() é o responsável por calcular o ponto de interseção do raio com o objeto mais próximo da câmera, chamar o método shade() e de fazer o loop para o cálculo da reflexão.

'vec3 trace(vec3 origin, vec3 direction)                                                                  \n' +

'{                                                                                                        \n' +

'   vec3 fragColor = vec3(0.0);                                                                           \n' +

'   vec3 percentage = vec3(1.0);                                                                    \n' +

'   for (float i = 0.0; i < ' + bounces + '; i++)                  \n' +

'   {                                                                                            \n' +

'       float t = ' + infinity + ';                                                                       \n' +

**Calcula o t da interseção com o objeto mais próximo**

'    if(t < ' + infinity + ')                                                                        \n' +

'       {                                                                                                 \n' +

'           bool isReflective = false;                                                                    \n' +

'           vec3 dcolor, scolor, normal;                                                                  \n' +

'           vec3 point = origin + t \* direction;                                                          \n' +

**Recupera a cor difusa e especular, e a normal do objeto no ponto de interseção**

'           vec3 color = shade(origin, direction, point, dcolor, scolor, normal);                         \n' +

'           if (isReflective)                                                                             \n' +

'           {                                                                                             \n' +

'               fragColor += percentage \* (vec3(1.0) - scolor) \* color;                      \n' +

'           }                                                                                             \n' +

'           else                                                                                          \n' +

'           {                                                                                             \n' +

'               fragColor += percentage \* color;                                                          \n' +

'               break;                                                                                    \n' +

'           }                                                                                             \n' +

'           vec3 toEye = normalize(point - origin);                                         \n' +

'           direction = reflect(toEye, normal);                                                     \n' +

'           origin = point;                                                                               \n' +

'           percentage \*= scolor;                                                                         \n' +

'       }                                                                                                 \n' +

'       else                                                                                              \n' +

'       {                                                                                                 \n' +'           fragColor += percentage \* ' + background + ';                                                 \n' +'           break;                                                                                        \n' +'       }                                                                                                 \n' +'   }                                                                                                  \n' +'   return fragColor;                                                                                     \n' +

'}                                                                                                        \n' +

Fundamentos da Computação Gráfica

Rendering

O método shade() que retorna a cor do pixel relativo as contribuições da luz ambiente, da componente difusa e da componente especular.

'vec3 shade(vec3 origin, vec3 direction, vec3 point, vec3 dcolor, vec3 scolor, vec3 normal)              \n' +

'{                                                 \n' +

'   vec3 toLight = vec3(0.0);                      \n' +

'   vec3 accumulatedColor = vec3(0.0);             \n' +

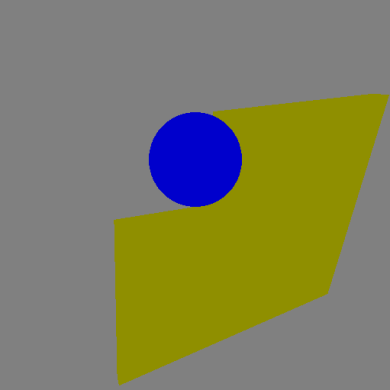
**Para cada fonte de luz calcula a componente difusa e especular e soma com a contribuição da luz ambiente**

'   return accumulatedColor;                       \n' +

'}                                                 \n' +

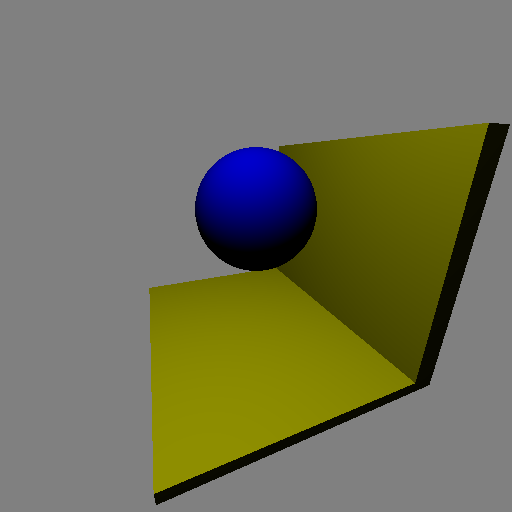
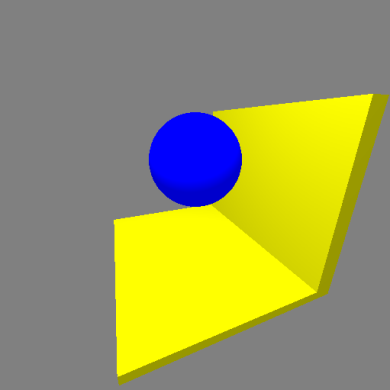
## Contribuição da Luz Ambiente

Abaixo podemos ver uma imagem da cena apenas com a contribuição da luz ambiente:



## Componente Difusa

Abaixo podemos ver duas imagens da cena. A primeira apenas da componente de reflexão difusa e a segunda com a contribuição da luz ambiente somada com a componente de reflexão difusa:

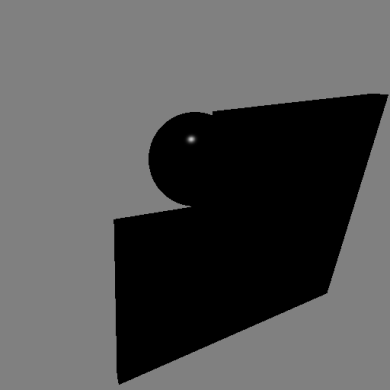
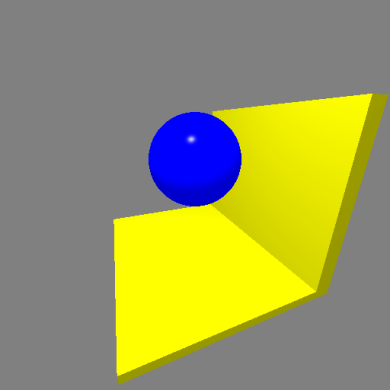
 

Fundamentos da Computação Gráfica

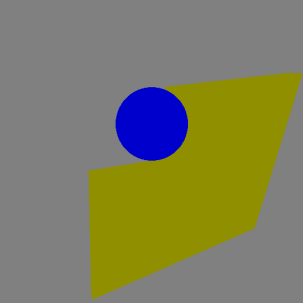
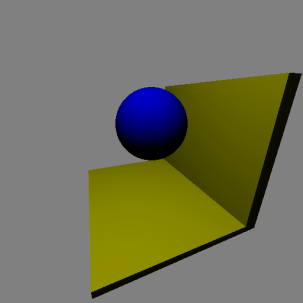
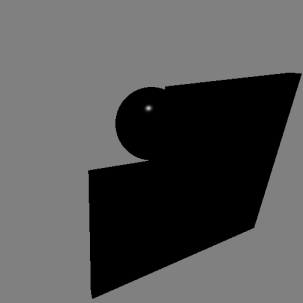
Rendering

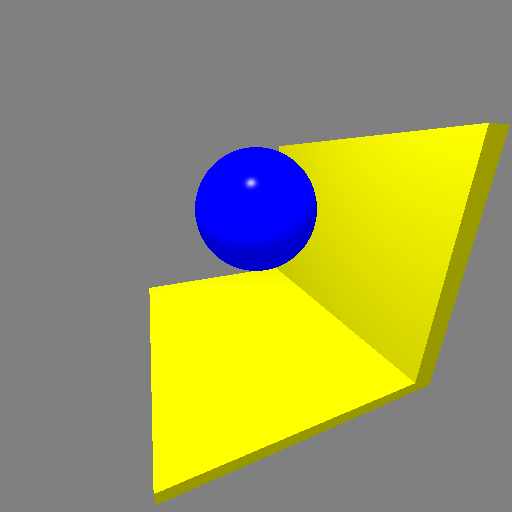
## Componente Especular

Abaixo podemos ver duas imagens da cena. A primeira apenas com a contribuição da componente de reflexão especular e a segunda com a contribuição da luz ambiente somada com a componente de reflexão difusa e com a componente de reflexão especular:

## Luz Ambiente + Componente Difusa + Componente Especular

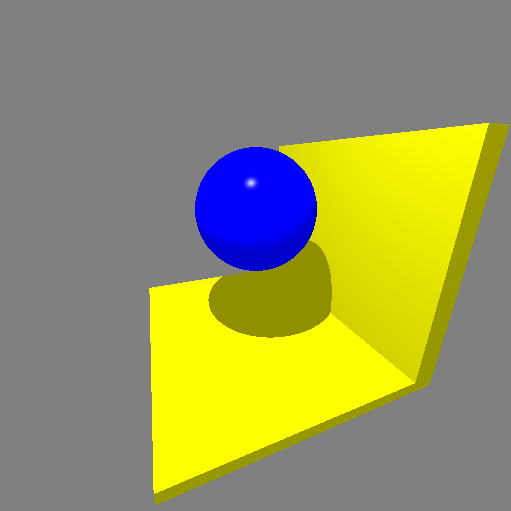


Fundamentos da Computação Gráfica

Rendering

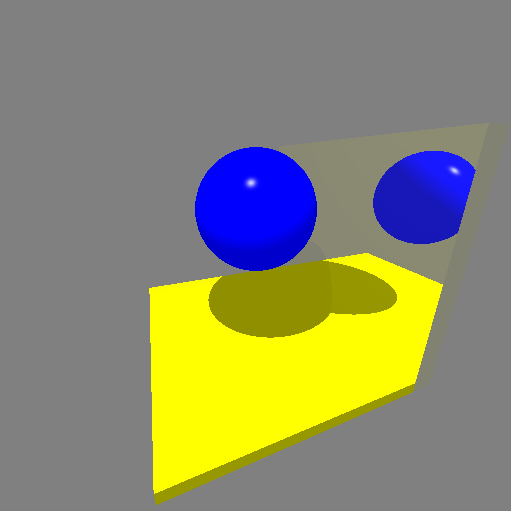
## Sombra

Abaixo podemos ver a imagem da cena com a contribuição da luz ambiente somada com a componente de reflexão difusa e com a componente de reflexão especular e também com o cálculo da sombra:



## Reflexão

Abaixo podemos ver a imagem da cena com a contribuição da luz ambiente somada com a componente de reflexão difusa e com a componente de reflexão especular, com o cálculo da sombra e também com o cálculo dos reflexos:



Fundamentos da Computação Gráfica

Rendering

## Mais Algumas Cenas

