UFMA

Departamento de Informática

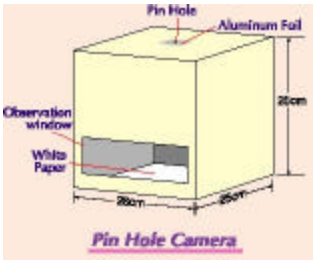
Introdução à Computação Gráfica
DEINF-UFMA
Prof. Anselmo Paiva

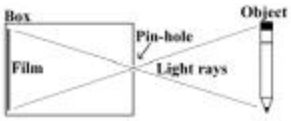
RayTracing
(Baseado nas Notas de Aula:
Computação Gráfica Interativa
Prof. Marcelo Gattass
TeCGraf- PUC/Rio)

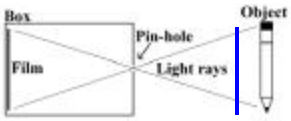
Pinturas: produção de imagens

Avercamp, Hendrick (1585-1634).

Pin-Hole Camera

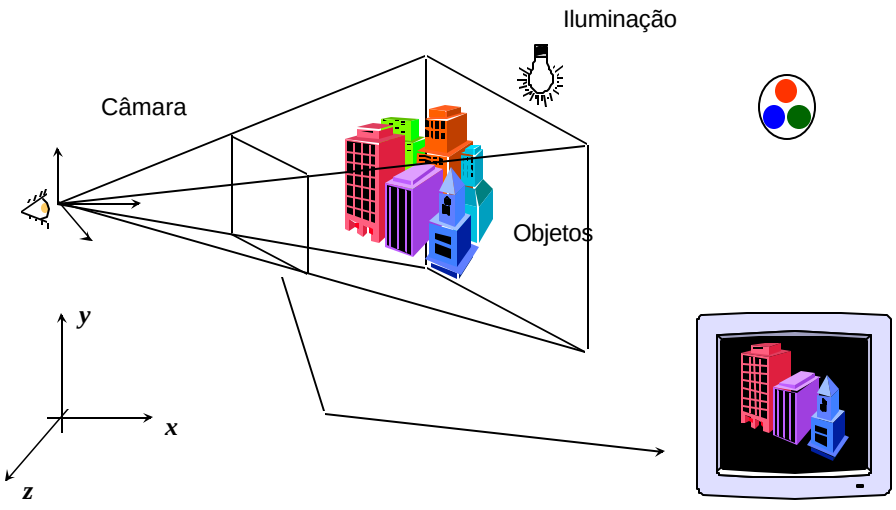




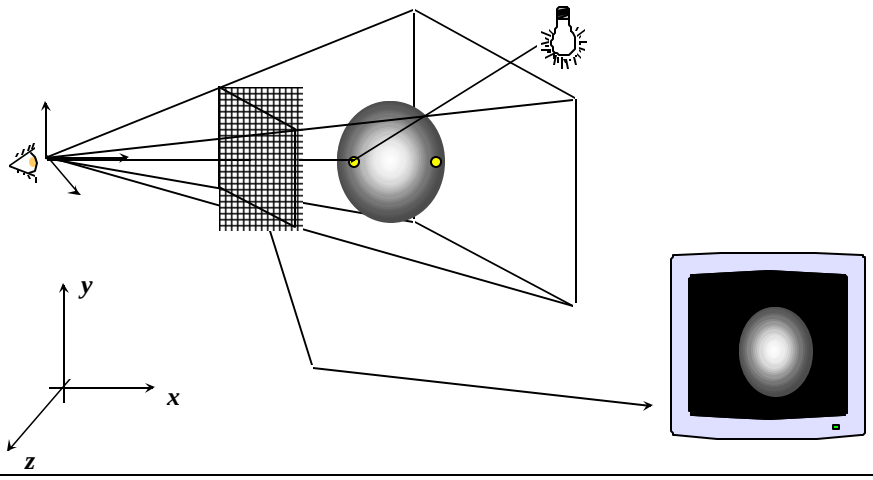


Câmara escura - Leonardo da Vinci -1545
Luis-Jacques-Mandé Daguerre (1789-1851)

Modelo de câmera

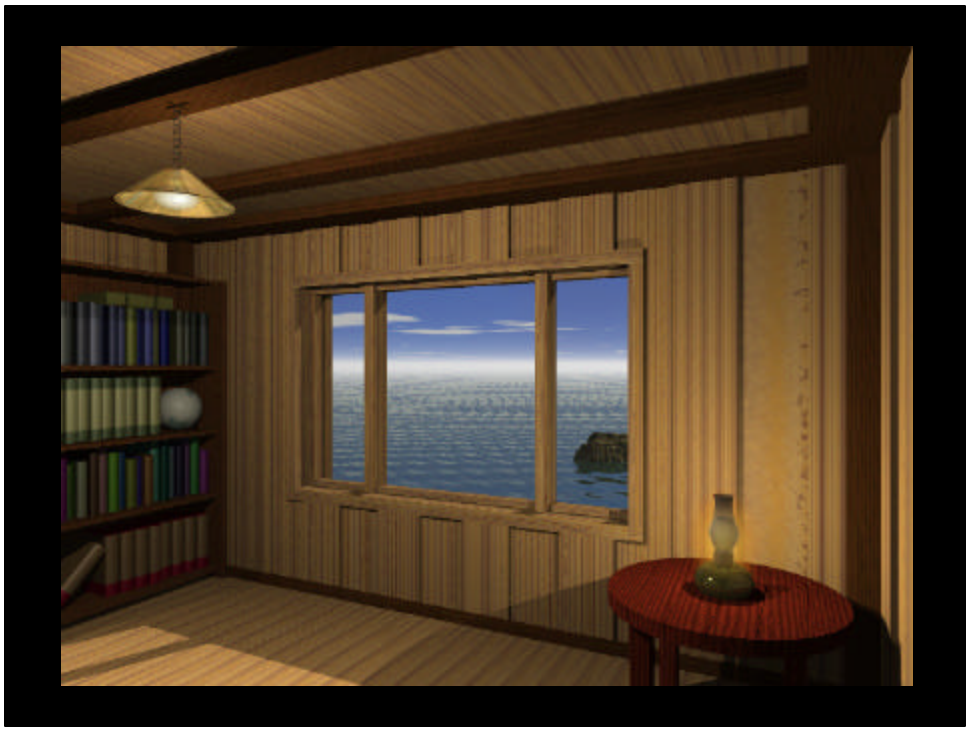


Traçado de raios

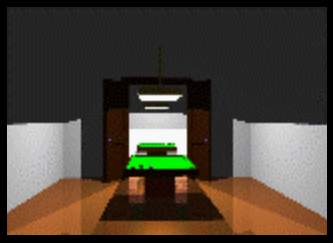


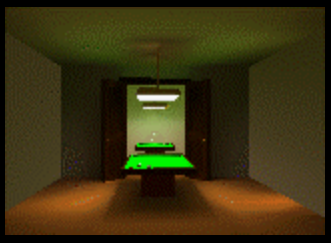
O Algoritmo Básico

```
/* define a lista de objetos, e as fontes de luz na cena */
/* define a posição do observador */
for ( j = 0; j < MAXROW; j++ )
  for ( i = 0; i < MAXCOL; i++ ) {
     /* lança o raio associado ao pixel i-j */
     /* constroi a equação do raio i-j */
     /* encontra todas as interseções do raio i-j com os objetos na cena */
     /* identifica o ponto de interseção mais próximo do observador */
     /* calcula a cor para esse pixel baseado no ponto de interseção e na posição da
        fonte de luz */
     /* salva a cor do pixel color ou desenha ele */
  }
}
```



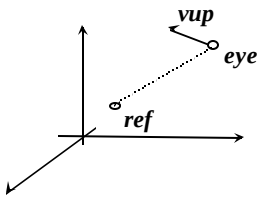

D. Scott Ferrie (C) 6/95



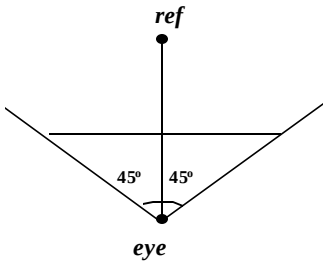


Modelo da câmera

Dados: ***eye***, ***ref***, ***vup*** *(definem o sistema de coordenadas do olho)*
abertura do campo fixa de 90^0

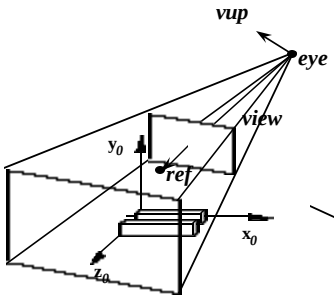


Coordenadas dos Objetos



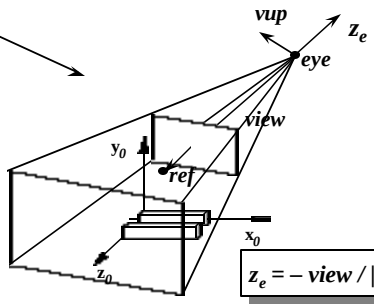
Vista de cima

Calcula o sistema - $x_e\ y_e\ z_e$



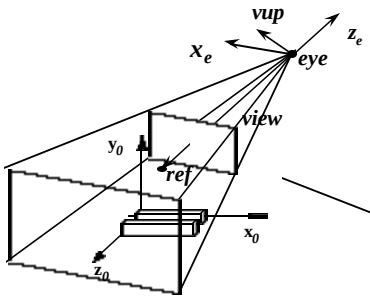
dados:
eye, ref, vup
abertura da câmera

view =ref-eye

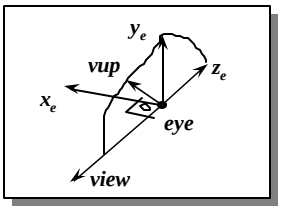


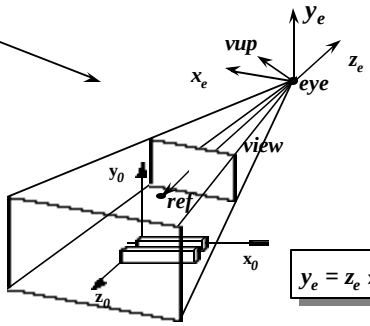
$z_e = -\ view\ /\ ||view||$

Calcula o sistema - $x_e\ y_e\ z_e$



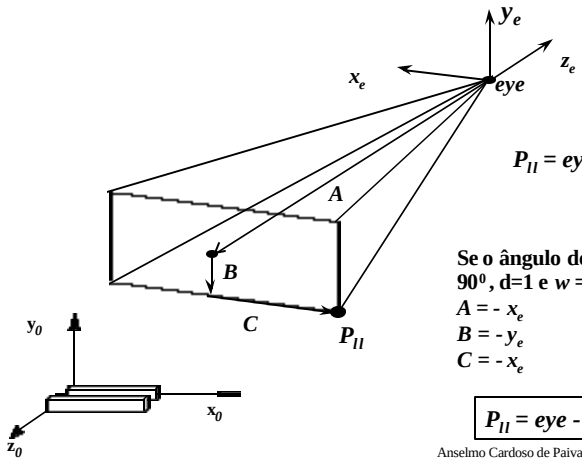
$$x_e = (vup \times z_e) / \|vup \times z_e\|$$





$$y_e = z_e \times x_e$$

Canto inferior da janela no plano de projeção



$$P_{ll} = eye + A + B + C$$

Se o ângulo de abertura for 90^0, d=1 e $w = h$:

$A = -x_e$

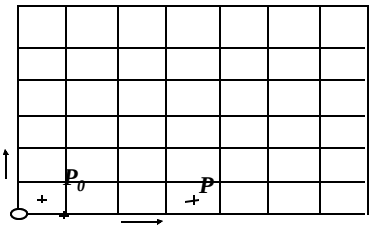
$B = -y_e$

$C = -x_e$

$$P_{ll} = eye - z_e - y_e - x_e$$

Lançamento de Raios

```
P0 = Pu + dx/2 + dy/2;
for (i=0; i<=w; i++)  {
   P = P0 ;
  for (j=0; j<=h; j++) {
      P = P + dx;
     trace o raio (eye, P);
   }
   P0 = P0 + dy;
}
```



$dy = 2\,y_e / h$

$dx = 2\,x_e / w$

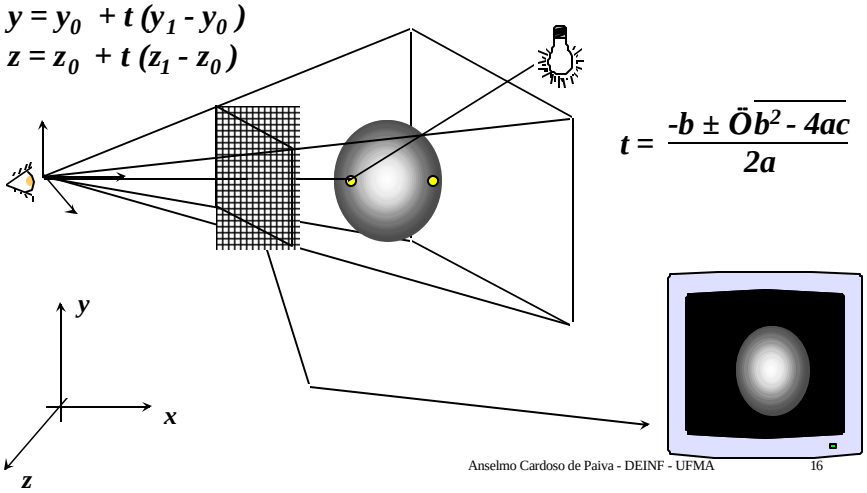
Traçado de raios: raios e objetos implícitos

Equação paramétrica(explícita) do raio

$$x = x_0 + t\,(x_1 - x_0)$$
$$y = y_0 + t\,(y_1 - y_0)$$
$$z = z_0 + t\,(z_1 - z_0)$$

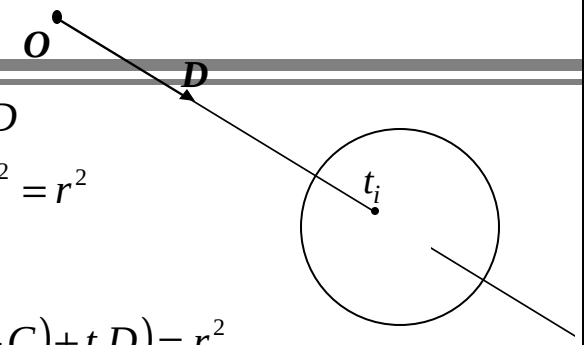
Equação implícita da esfera

$$(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 = R^2$$



$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Interseção com a esfera



$$Raio: \quad P(t) = O + tD$$

$$Esfera: \quad \|P(t_i) - C\|^2 = r^2$$

$$\|O + t_i D - C\|^2 = r^2$$

$$((O-C) + t_i D) \cdot ((O-C) + t_i D) = r^2$$

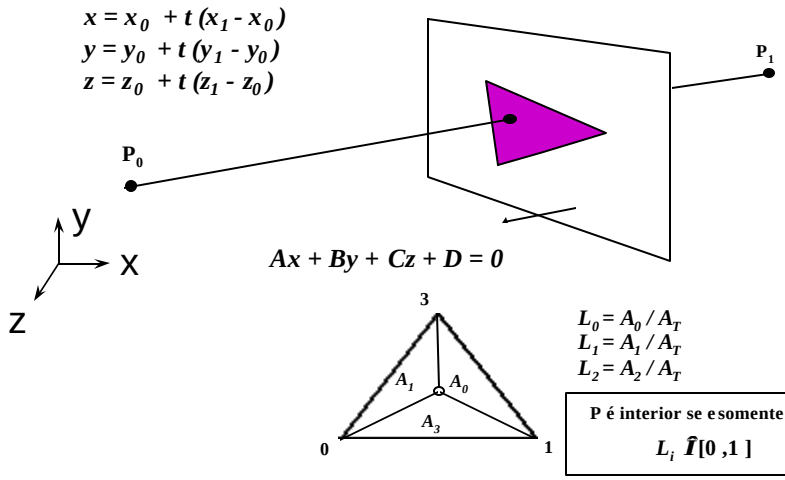
$$[D \cdot D]\, t_i^2 + [2D \cdot (O-C)]\, t_i + [(O-C) \cdot (O-C) - r^2] = 0$$

$$a\, t_i^2 + b\, t_i + c = 0$$

As raizes t_0 e t_1 são examinadas, escolhendo-se a mais próxima

int = (x_0+x_d*t, y_0+y_d*t, z_0+z_d*t)

Raios e objetos descritos pela fronteira



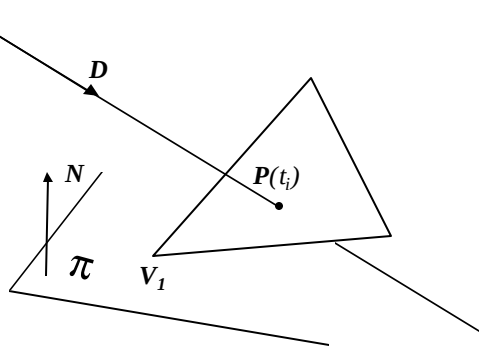
$L_0 = A_0 / A_T$
$L_1 = A_1 / A_T$
$L_2 = A_2 / A_T$

P é interior se e somente se

$L_i \in [0, 1]$

Interseção com o plano do triângulo

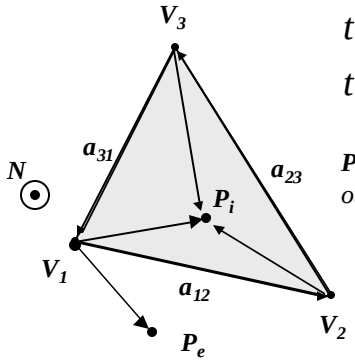
$$P \in p \leftrightarrow (P - V_1) \cdot N = 0$$
$$(O + t_i D - V_1) \cdot N = 0$$
$$t_i D \cdot N + (O - V_1) \cdot N = 0$$



$$t_i = \frac{(V_1 - O) \cdot N}{D \cdot N}$$

$$P = O + t_i D$$

Ponto no interior de um triângulo



$$t_1 = a_{12} \times (P - V_1)$$
$$t_2 = a_{23} \times (P - V_2)$$
$$t_3 = a_{31} \times (P - V_3)$$

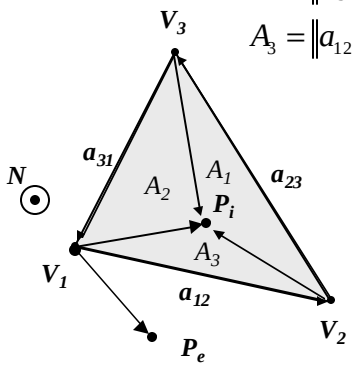
***P** é interior se **t_1**, **t_2** e **t_3** tem o mesmo sentido, ou seja:*

$$(t_1 \cdot t_2) > 0 \quad (t_1 \cdot t_2) < 0$$
$$(t_1 \cdot t_3) > 0 \quad \text{ou} \quad (t_1 \cdot t_3) < 0$$

Coordenadas baricêntricas

$$A_1 = \|a_{23} \times (P - V_1)\| / 2 \qquad L_1 = A_1 / A_T$$
$$A_2 = \|a_{31} \times (P - V_2)\| / 2 \qquad L_2 = A_2 / A_T$$
$$A_3 = \|a_{12} \times (P - V_3)\| / 2 \qquad L_3 = A_3 / A_T$$

$$A_T = A_1 + A_2 + A_3$$

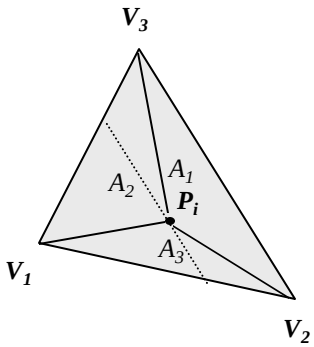


***P** é interior se L_1, L_2 e $L_3 \in$ [0..1]*

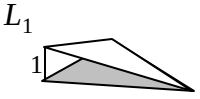
Mais ainda:

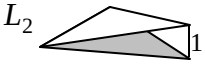
$$\begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} = L_1 \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} + L_2 \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} + L_3 \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix}$$

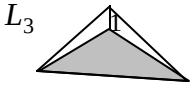
Coordenadas baricêntricas como interpolantes



$$\begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} = L_1 \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} + L_2 \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} + L_3 \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix}$$

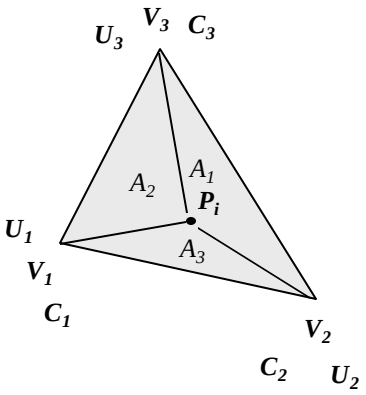






$$L_1 + L_2 + L_3 = 1$$

Interpolação com coordenadas baricêntricas

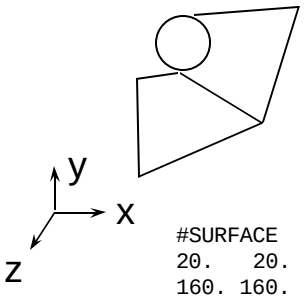


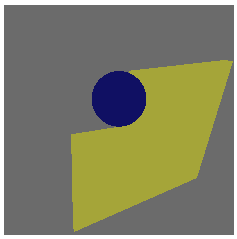
$$\begin{Bmatrix} r_i \\ g_i \\ b_i \end{Bmatrix} = L_1 \begin{Bmatrix} r_1 \\ g_1 \\ b_1 \end{Bmatrix} + L_2 \begin{Bmatrix} r_2 \\ g_2 \\ b_2 \end{Bmatrix} + L_3 \begin{Bmatrix} r_3 \\ g_3 \\ b_3 \end{Bmatrix}$$

ou:

$$\begin{Bmatrix} u_i \\ v_i \end{Bmatrix} = L_1 \begin{Bmatrix} u_1 \\ v_1 \end{Bmatrix} + L_2 \begin{Bmatrix} u_2 \\ v_2 \end{Bmatrix} + L_3 \begin{Bmatrix} u_3 \\ v_3 \end{Bmatrix}$$

Exemplo: uma esfera e duas paredes





```
#SURFACE
20.   20.  100.
160. 160.  60.
#LIGHT
40. 120. 0.     255 255 255
#SPHERE
0  25.0     0.0  20.0  0.0
#BOX
1   -80. -50.  -50.   50. -45.  50.
1   -80. -50.  -60.   50. 50.  -50.
```

Programa Exemplo

Main

```
void main( void )
{
  int linhas,colunas;   // número de linhas e colunas da imagem
  char arquivo[255];    // entrada:  arquivo.dat, saída: arquivo.ppm
  Cor_rgb matrix[MAX_RES*MAX_RES]; // matrix de cores

  int i, j, k=0;  // contadores para linhas, colunas e posição no vetor de cores, respectivamente

 //* Objetos:
  Camera camera;            // cria um objeto câmera default
  Cenario cenario;          // cria um objeto cenário vazio
  Raio raio;                // cria um objeto raio default

 //* Código:
  if(!LeArquivo( &cenario, &camera, &linhas, &colunas, arquivo)) return;
  raio = camara.PrimeiroRaio();  //  Pega o primeiro raio
  for(i=0;i<linhas;i++) {
    for(j=0;j<colunas;j++) {
      matrix[k++] = cenario.Intercepta( raio, 0 );
      raio = camera.ProximoRaio();
    }
  }
  SalvaPPM( linhas, colunas, 255, matrix, arquivo);
}
```

Vector_3D

```
class Vetor_3D{
  private:
    float x, y, z;

  public:
  //* Construtores e Destrutor
    Vetor_3D() : x(0.0), y(0.0), z(0.0) {};
    Vetor_3D(float _x, float _y, float _z) : x(_x), y(_y), z(_z) {};
    ~Vetor_3D() {};
  //* Atribuições dos valores de x y z
    void Atribui(float _x, float _y, float _z);
    void Copia(Vetor_3D origem);
  //* Operações sobre vetor
    void Soma(Vetor_3D origem);
    void Subtrai(Vetor_3D origem);
    void Divide(float divi);
    void Multiplica(float multi);
    void Negativo();
    void Normaliza();
  //* Funções que retornam alguma informação sobre o vetor
    float Comprimento();
    float ProdutoEscalar(Vetor_3D outro);
    Vetor_3D ProdutoVetorial(Vetor_3D outro);
  //* Funções inline  que retornam os valores de x y z
    inline float Vetor_3D::X()  {  return(x); };
    inline float Vetor_3D::Y()  {  return(y); };
    inline float Vetor_3D::Z()  {  return(z); };
};
```

```
void Vetor_3D::Soma(Vetor_3D origem)
{
  x += origem.x;
  y += origem.y;
  z += origem.z;
}
```



```
class Raio{

  private:
    Vetor_3D origem, destino, direcao;
  public:
  //* Construtores e Destrutor
    Raio();
    Raio(Vetor_3D _origem, Vetor_3D _destino);
    ~Raio() {};

  //* Atribuição dos vetores origem e destino
    void  Atribui(Vetor_3D _origem, Vetor_3D _destino);

  //* Funções inline  que retornam informações do Raio
    inline Vetor_3D Origem()  {return origem;};
    inline Vetor_3D Destino() {return destino;};
    inline Vetor_3D Direcao() {return direcao;};
    inline float    Dx()      {return direcao.X();};
    inline float    Dy()      {return direcao.Y();};
    inline float    Dz()      {return direcao.Z();};
    inline float    X0()      {return origem.X();};
    inline float    Y0()      {return origem.Y();};
    inline float    Z0()      {return origem.Z();};

  //* QualPonto  retorna o ponto x y z para um t da função paramétrica
    Vetor_3D QualPonto(float t);
};
```

```
class Camera{

  private:
    Vetor_3D P0, dx, dy, P_atual, P_linha, olho, direcao, vup, xe, ye, ze;
    int linhas, colunas, l_atual, c_atual;

    void PreparaRaios( ); // Esta função _Private_ calcula a câmera e prepara os raios


  public:
          Camera();          //* Construtores e Destrutor
          ~Camera();
    void Atribui( Vetor_3D olho, Vetor_3D _direcao, Vetor_3D _vup,
                  int _linhas, int _colunas );

    Raio PrimeiroRaio(); //* Pegando os Raios
    Raio ProximoRaio();
};
```

```
Camera::Camera( )
{
  linhas = 200;
  colunas = 200;
  olho.Atribui( 0.0, 0.0, 0.0);
  direcao.Atribui( 0.0, 0.0, -1.0 );
  vup.Atribui(0.0, 1.0, 0.0 );

  PreparaRaios();
}
```



```
bool LeArquivo( Cenario * hcenario, Camera * hcamera,
                int *hlins, int *hcols, FILE * arqdat)
{
 tok  tag; char linha[162]; Vetor_3D vets[6];

 fgets( linha, 7, arqdat );
 if(strcmp(linha,"RT 1.0"))   return false;
 FimLinha(arqdat);   //primeira linha

 //Loop para leitura das linhas do arquivo
 tag = tok_DESCONHECIDO;
 while(fgets( linha, 161, arqdat )!=NULL) {
    if (linha[0]=='#')
      tag = Token(&linha[1]);   //já dispenso a #
    else
      if (linha[0]!=';' && linha[0]!='!') // comentários
        LeInfo( tag, hcenario, hcamara, hlins, hcols, linha );
 }

fclose(arqdat);
return true;
}
```

```
//Dispensa fim da linha
void FimLinha(FILE * arqdat)
{
  char ch;
  while((ch=fgetc(arqdat))!=10);
}
```

```
//Reconhece o token
tok Token(char linha[82])
{
  linha[10]=0;
  if(!strcmp(linha,"BACKGROUND")) return tok_BACKGROUND;

  linha[8]=0;
  if(!strcmp(linha,"POSITION")) return tok_POSITION;

  if(!strcmp(linha,"TRIANGLE")) return tok_TRIANGLE;

  linha[7]=0;
  if(!strcmp(linha,"SURFACE")) return tok_SURFACE;

  linha[6]=0;
  if(!strcmp(linha,"SPHERE")) return tok_SPHERE;

  linha[5]=0;
  if(!strcmp(linha,"LIGHT")) return tok_LIGHT;

  linha[4]=0;
  if(!strcmp(linha,"SIZE")) return tok_SIZE;

  linha[3]=0;
  if(!strcmp(linha,"BOX")) return tok_BOX;

  return tok_DESCONHECIDO;
}
```

```
typedef enum {
  tok_DESCONHECIDO = 0,
  tok_SIZE,
  tok_BACKGROUND,
  tok_SURFACE,
  tok_LIGHT,
  tok_SPHERE,
  tok_BOX,
  tok_TRIANGLE,
  tok_POSITION
}tok;
```



```
void LeInfo( tok tag, Cenario *hcenario, Camara *hcamara,
             int *hlinhas, int *hcolunas, char *linha )
{
  int a,b,c; float e, f, g; Cor_rgb cor1,cor2,cor3;
  Vetor_3D vet1, vet2, vet3, vets[3];
  Material *mat1; Luz *luz1; Esfera *esf1; CaixaParalela *cxp1; Triangulo *tri1;
  int v=0;

  switch(tag)
  {
    case tok_SIZE:
      sscanf(linha,"%d %d",hlinhas,hcolunas);
      *hlinhas = MINI(*hlinhas,MAX_RES);       // Para não estourar a minha
      *hcolunas = MINI(*hcolunas,MAX_RES);     // matriz[MAX_RES*MAX_RES]
      return;

    case tok_BACKGROUND:
      cor1.Copia(LeCor(linha));
      hcenario->RecebeFundo(cor1);
      return;

   ...

    case tok_DESCONHECIDO:
      return;
  }
}
```

```
Cor_rgb LeCor(char *linha)
{
  Cor_rgb cor;
  int a,b,c;

  sscanf(linha,"%d. %d. %d.",&a,&b,&c);
  cor.Atribui(a,b,c);
  return cor;
}
```

```
case tok_SURFACE:
  cor1.Copia(LeCor(linha));      // Ia =Luz ambiente
  v = Dispensa(linha,3);
  cor2.Copia(LeCor(&linha[v])); // Kd = Cor da difusa
  v += Dispensa(&linha[v],3);
  cor3.Copia(LeCor(&linha[v])); // Ks =Cor da especular
  v += Dispensa(&linha[v],3);
  sscanf(&linha[v],"%d. %f %f %f",&a,&e,&f,&g);
  mat1 = new Material( cor1, cor2, cor3, a, e, f, g );
  hcenario->InsereMaterial( mat1 );
  return;

case tok_LIGHT:
  vet1.Copia(LeVetor(linha));   //Posição
  v = Dispensa(linha,3);
  sscanf(&linha[v],"%d %d %d",&a,&b,&c);
  //Esta cor não pode ser lida como as demais por não ter '.'
  cor1.Atribui(a, b, c);    //Cor
  luz1 = new Luz( vet1, cor1 );
  hcenario->InsereLuz( luz1 );
  return;
```

```
int Dispensa(char *linha, int quantos)
{
  int i=0;
  while (quantos>0) {
    i++;
    if (linha[i]==' ') {
      quantos--;
      while (linha[++i]==' ');
    }
  }
  return i;
}
```



```
case tok_SPHERE:
  sscanf(linha,"%d",&a);
  v = Dispensa(linha,1);
  sscanf(&linha[v],"%f",&e);       //Raio
  v += Dispensa(&linha[v],1);
  vet1.Copia(LeVetor(&linha[v])); //Centro
  esf1 = new Esfera( a, e, vet1 );
  hcenario->InsereObjeto( esf1 );
  return;

case tok_BOX:
  sscanf(linha,"%d",&a);
  v = Dispensa(linha,1);
  vet1.Copia(LeVetor(&linha[v])); //canto inferior esquerdo
  v += Dispensa(&linha[v],3);
  vet2.Copia(LeVetor(&linha[v])); //canto superior direito
  cxp1 = new CaixaParalela( a, vet1, vet2 );
  hcenario->InsereObjeto( cxp1 );
  return;
```

```
case tok_TRIANGLE:
  sscanf(linha,"%d",&a);
  v = Dispensa(linha,1);
  vets[0].Copia(LeVetor(&linha[v])); //primeiro vértice
  v += Dispensa(&linha[v],3);
  vets[1].Copia(LeVetor(&linha[v])); //segundo vértice
  v += Dispensa(&linha[v],3);
  vets[2].Copia(LeVetor(&linha[v])); //terceiro vértice
  tri1 = new Triangulo( a, vets );
  hcenario->InsereObjeto( tri1 );
  return;


case tok_POSITION:
  vet1.Copia(LeVetor(linha));        //eye
  v = Dispensa(linha,3);
  vet2.Copia(LeVetor(&linha[v]));    //ref
  v += Dispensa(&linha[v],3);
  vet3.Copia(LeVetor(&linha[v]));    //up
  hcamara->Atribui( vet1, vet2, vet3, *hlinhas, *hcolunas );
  return;
```


Exemplo de Arquivo de Cena

```
RT 1.0
#SIZE
500 500          ! w e h
#BACKGROUND
20. 20. 40.      ! cor de Fundo
#SURFACE
0.   0.  50.     ! Ia*Ka (=luz ambiente*Ka)
0.   0. 255.     ! Kd
255.  255.  255. ! Ks
50.              ! n
0.               ! coef. de reflexao  (multiplica o resultado do raio Refletido)
0.               ! indice de refracao do material,se opacidade <1 (lei de Snell)
1.               ! opacidade  (0 e' transparente e 1 e' opaco)
30. 30.   0.    120. 120.   0.    255.  255.  255.   40.   0.  0.  1.
#LIGHT
40. 120. 0.      ! posicao
255 255 255      ! intensidade rgb
```

Exemplo de Arquivo de Cena (cont.)

```
#SPHERE
0                ! tipo de superficie
25.0             ! raio
0.0  20.0  0.0   ! centro
#BOX
1                ! tipo de superficie
-80. -50.  -50.  ! canto inferior esquerdo
 50. -45.  50.   ! canto superior direito
#FILM
35. 35.          ! parametro de lente (ignore)
#POSITION
100. 40. 40.     ! posicao do eye
0. 0. 0.         ! posicao do ref
0. 1. 0.         ! vetor up
#GRID
1 1              ! nivel de sub-pixels (ignore)
#LENS
20.              ! profundidade de campo (ignore, use uma abertura de camera de
                   90 graus na horizontal)
```


Trabalho 3 - Primeira Parte

Raycasting

"Implementar as rotinas de interseção do raio com os objetos previstos no programa, e gerar uma imagem que desenhasomente a cor do objeto interceptado que está mais próximo do observador"

Modelos de Iluminação

- Durante a interação da luz com um objeto, parte da energia é absorvida, parte é transmitida e parte é refletida na superfície do objeto.
- A componente refletida da energia luminosa incidente
- é a responsável pela sensação de cor produzida no cérebro de um ser humano.
- A quantidade de luz refletida depende da:
  - composição, direção e geometria da fonte de luz;
  - orientação da superfície do objeto em relação à fonte de luz;
  - propriedades da superfície do objeto.

Modelo de Bouknight

- reflexão da luz:
  - especular;
  - difusa.
- Lei de Lambert:

$I=I_l k_d \cos(\Theta)$;

$0 * \Theta * \Pi/2;\ 0 * k_d * 1$

I_l é a intensidade da luz incidente

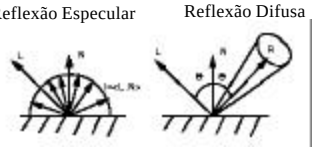
k_d é um coeficiente de atenuação (car. material)

- uma fonte de luz pontual - pontos que não recebem luz diretamente da fonte aparecem pretos.
- Cenas reais - os objetos recebem luz de forma indireta, a partir de reflexões no ambiente - termo difuso:

$I=I_a k_a+I_l k_d \cos(\Theta)$;

I_a representa a intensidade da luz ambiente

Reflexão Especular Reflexão Difusa



Lei de Lambert

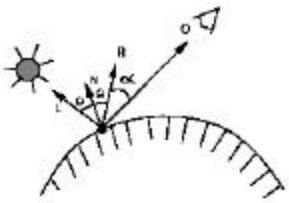
Modelo despreza componente refletida de forma especular.

Modelo de Phong

- Extensão empírica, para simular a reflexão especular.
- Termo introduzido:

$I_e=I_l k_e \cos^n(\alpha)$; $0 * k_e * 1$

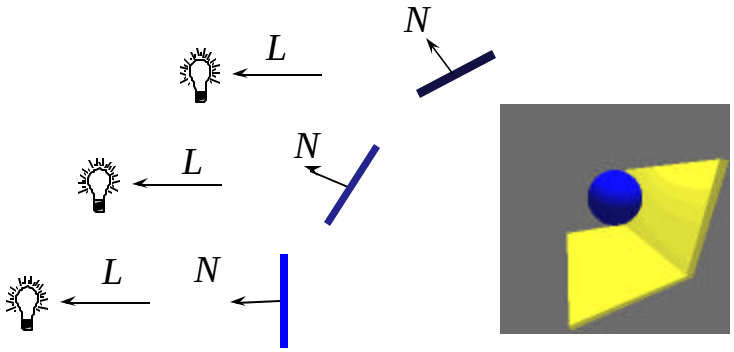
α é o ângulo entre o raio refletido e a direção de observação.



- Modelo de iluminação local:

a intensidade luminosa em um ponto depende apenas das fontes diretas de luz.

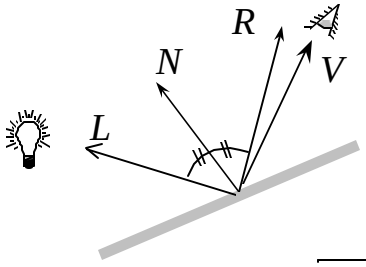
Componente de reflexão difusa



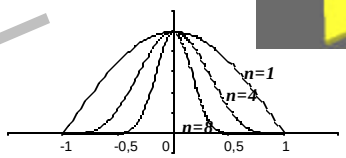
$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} (N \bullet L) = \begin{pmatrix} l_r \cdot k_{dr} \\ l_g \cdot k_{dg} \\ l_b \cdot k_{db} \end{pmatrix} (N \bullet L)$$

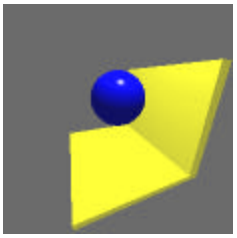
I, l, k **Î [0 ,1]**

Componente de reflexão especular



Brilho

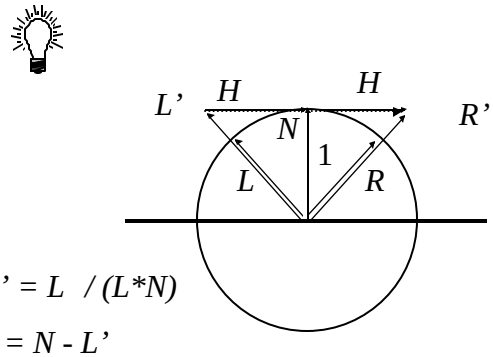




$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} (R \bullet V)^n = \begin{pmatrix} l_r \cdot k_{sr} \\ l_g \cdot k_{sg} \\ l_b \cdot k_{sb} \end{pmatrix} (R \bullet V)^n$$

I, l, k **Î [0 ,1]**

Cálculo do vetor R



$L' = L / (L*N)$

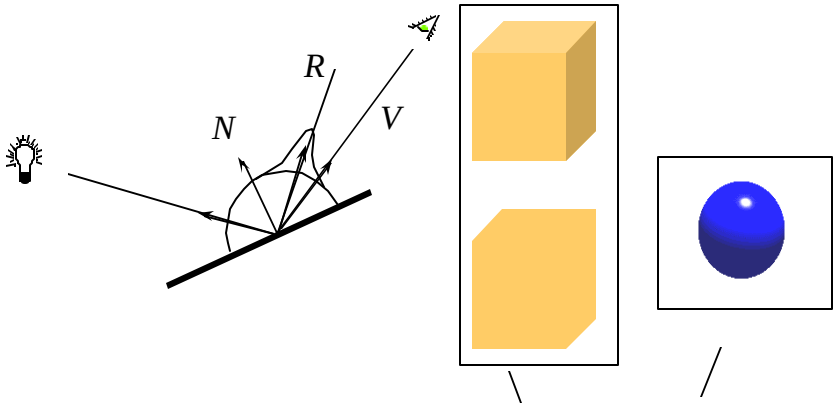
$H = N - L'$

$R' = N + H$

$$\vec{R} = 2\vec{N} - \frac{1}{(\vec{L} \bullet \vec{N})}\vec{L}$$

$R = unitário\ (R')$

Luz direta sobre um ponto



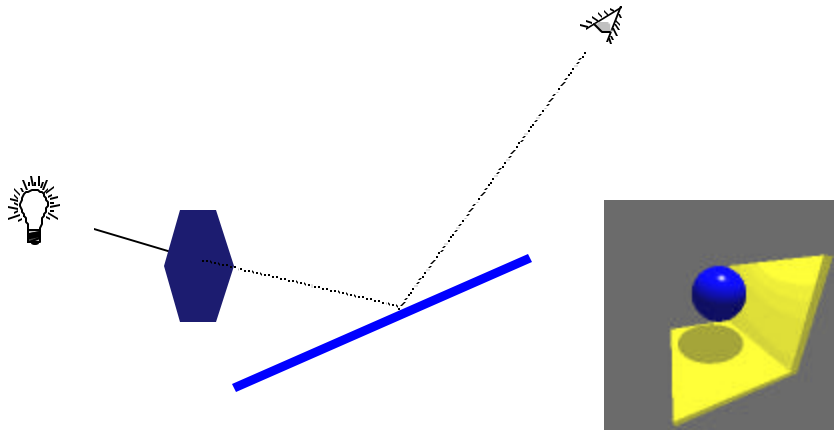
$$C_l = C_{amb_l} + C_{luz_l} k_{dif\ l} (N \bullet L) + k_s (R \bullet V)^n$$

Modelo de várias luzes

$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix} + \sum_{luzes} \left(\begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} (N \bullet L) + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} (R \bullet V)^n \right)$$

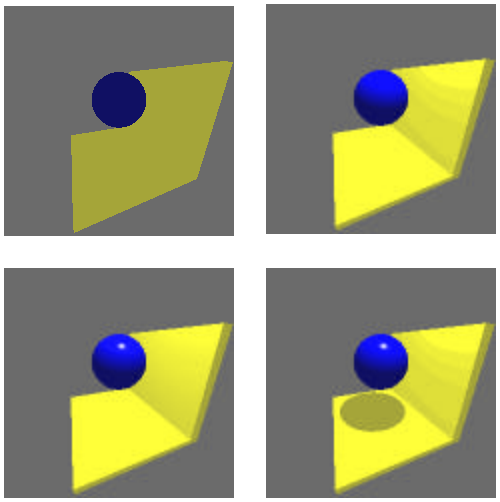
$$\begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix} = \textbf{Luz Ambiente}$$

Sombra



A luz não chega a superfíce

Uma revisão



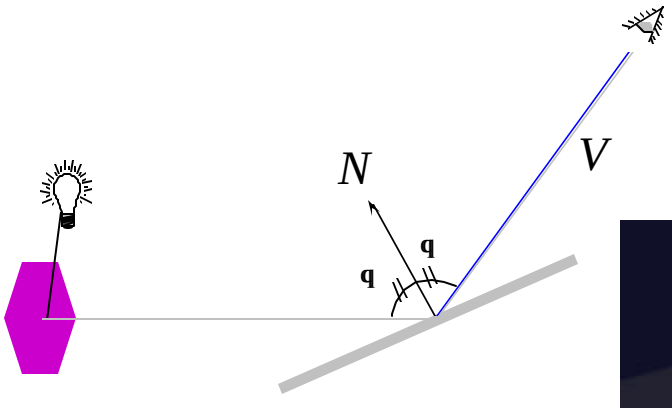

```
rtColor shade ( rtObject object, rtRay ray, int depth,
                rtPoint point, rtNormal normal, int depth)
{
 rtColor color, rColor, sColor;
 rtRay rRay, tRay, sRay;

color = termo ambiente;
for (cada luz) {
  sRay = raio para o ponto de luz;
  if (sRay • normal>0) {
     calcule quanto de luz é bloqueada por superfícies opacas e
     transparentes e use para computar as componentes difusa e especular } }

if (depth >= maxDepth) return color;

if (objeto é refletor) {
   rRay = raio na direção de refleção;
   rColor = trace( rRay, depth+1);
   reduza rColor pelo coeficente de refleção especular e some a color; }

if (objeto é transparente) {
   tRay = raio na direção de refração;
   if (reflexão total não ocorre) {
      tColor = trace( tRay, depth+1);
      reduza tColor pelo coeficente de refração especular e some a color; } }

return color;
}
```

Trabalho 3 - Segunda Parte

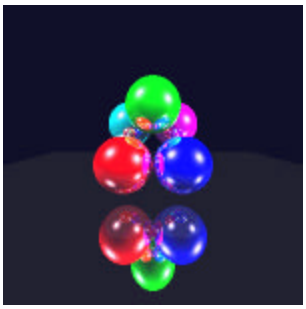
RayTracing

"Implementar as rotinas do modelo de iluminação de Phong, no Programa Ray Tracing"

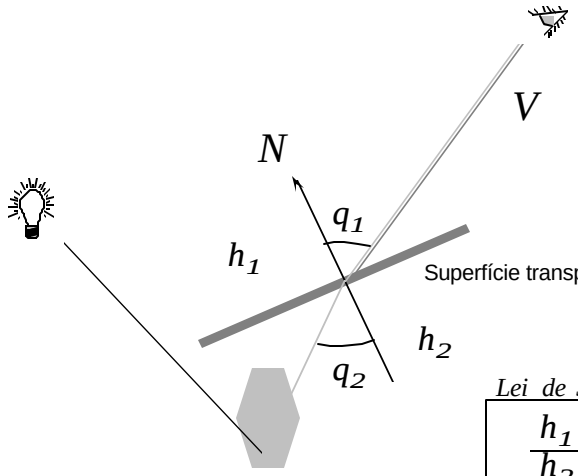
Reflexão de outros objetos



Superfície especular



Transparência



Lei de Snell:

$$\frac{h_1}{h_2} = \frac{\sin q_2}{\sin q_1}$$

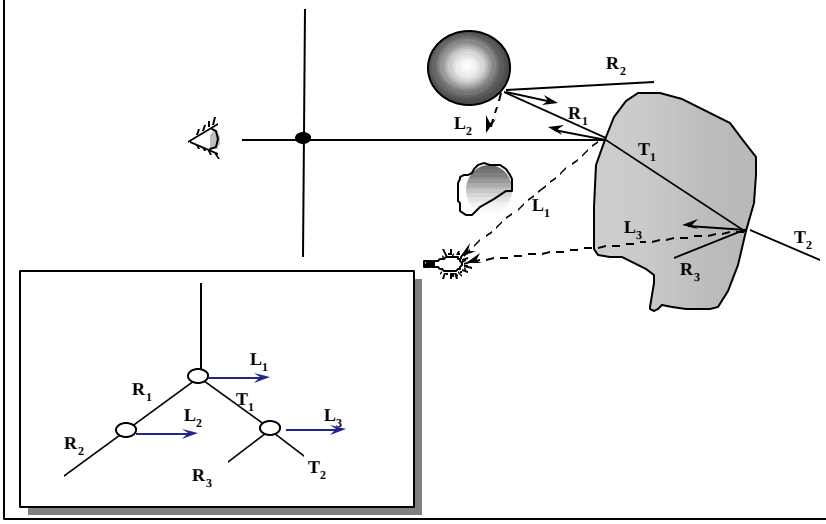
Iluminação considerando R e T

$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix} + \sum_{luzes} \left(\begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} (N \bullet L) + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} (R \bullet V)^n \right) + c_r \cdot \begin{pmatrix} I_r(R) \\ I_g(R) \\ I_b(R) \end{pmatrix} + (1-o) \cdot \begin{pmatrix} I_r(T) \\ I_g(T) \\ I_b(T) \end{pmatrix}$$

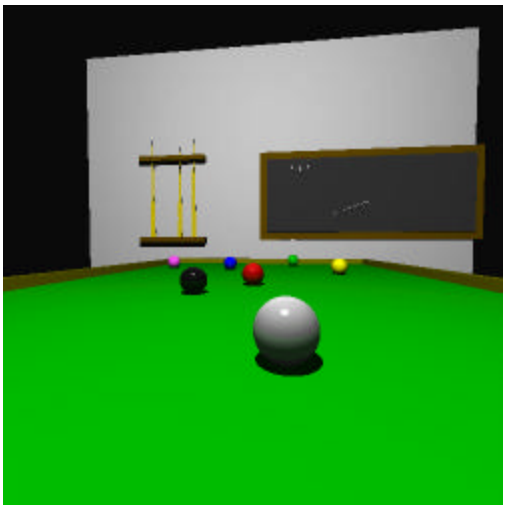
redução da reflexão

transparência

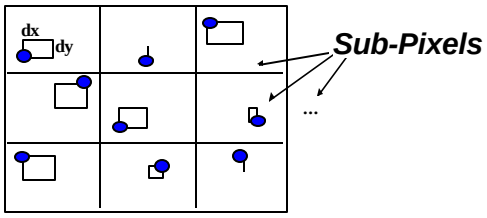
Traçado de Raios Recursivo



Resultado Esperado



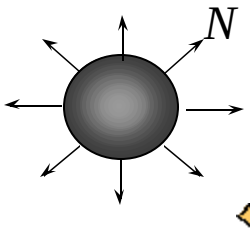
Tratamento anti-alias



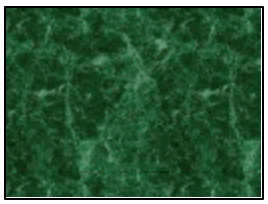
dx, dy = variáveis randômicas

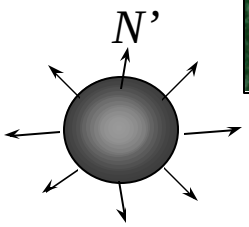
- **Lance um raio para cada sub-*pixel***
- **Faça uma média dos valores obtidos**

Normal e cor



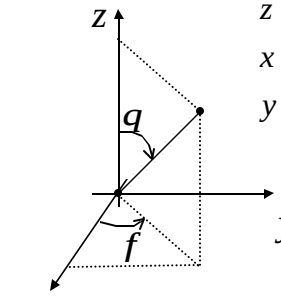
A cor de um ponto depende de sua posição num mapa de textura





Pertubar aleatóriamente as normais dos objetos

Textura na esfera

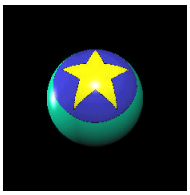


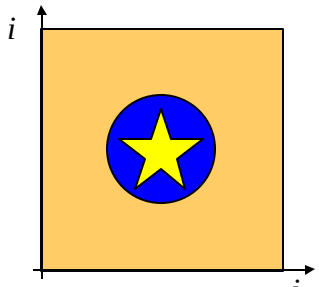
$$z = r\cos\theta$$
$$x = r\sin\theta\cos\phi$$
$$y = r\sin\theta\sin\phi$$

$$\tan\phi = y/x$$
$$\tan\theta = \frac{\sqrt{x^2 + y^2}}{z}$$

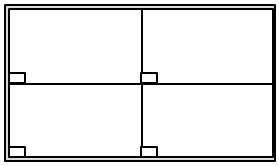
$$u = 1 + \phi/\pi$$
$$v = 1 - \theta/\pi$$

$$i = v(h-1)$$
$$j = u(w-1)$$

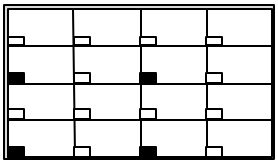




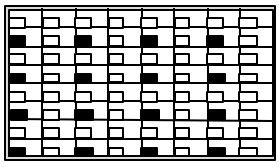
Refinamento Progresivo



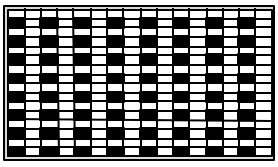
amostragem inicial



primeira subdivisão



segunda subdivisão



subdivisão final

pixels sendo visitados

pixels já visitados