Computação Gráfica

Visão Tridimensional

José Luis Seixas Junior

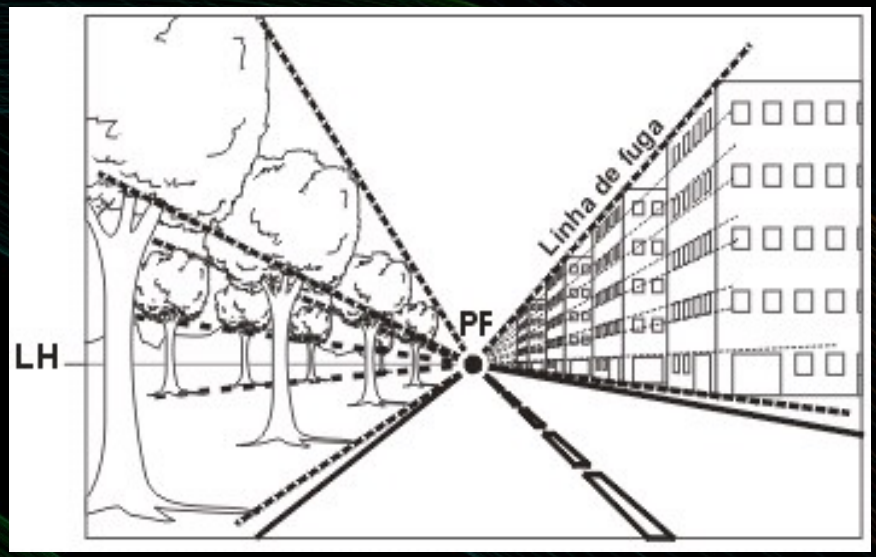
Índice

- Introdução;
- Câmera em perspectiva;
- Movimentos de câmera;
- Classe Camera;

Introdução

- Para criar maior realismo em cenas tridimensionais, devemos utilizar o modelo de projeção em perspectiva;

Introdução

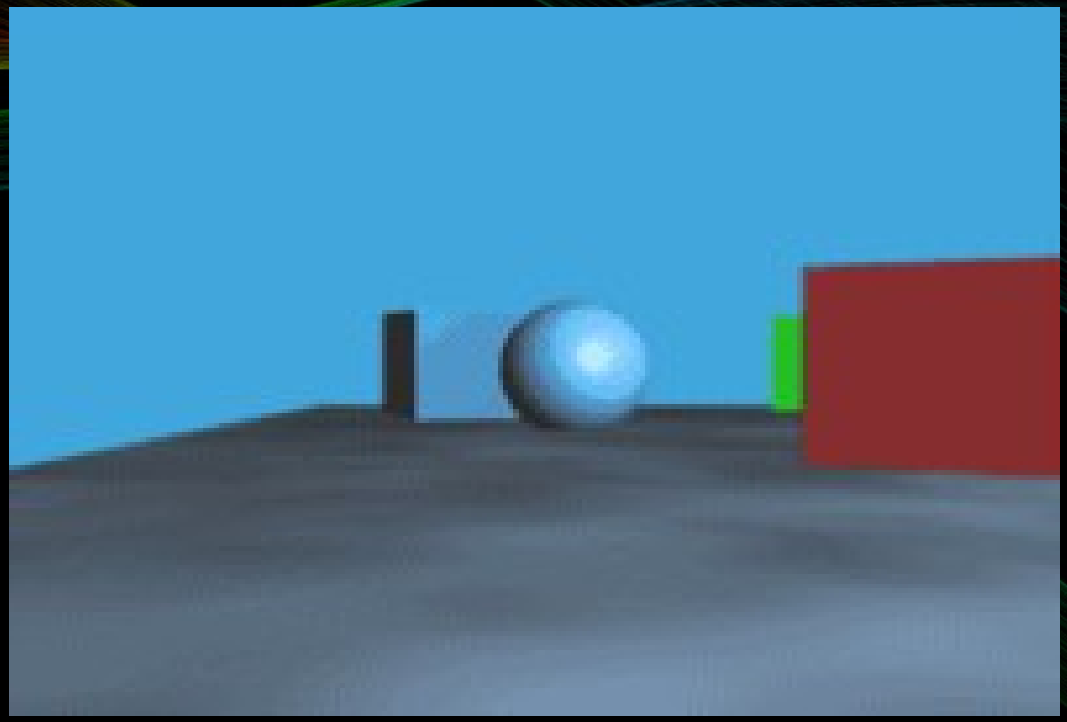


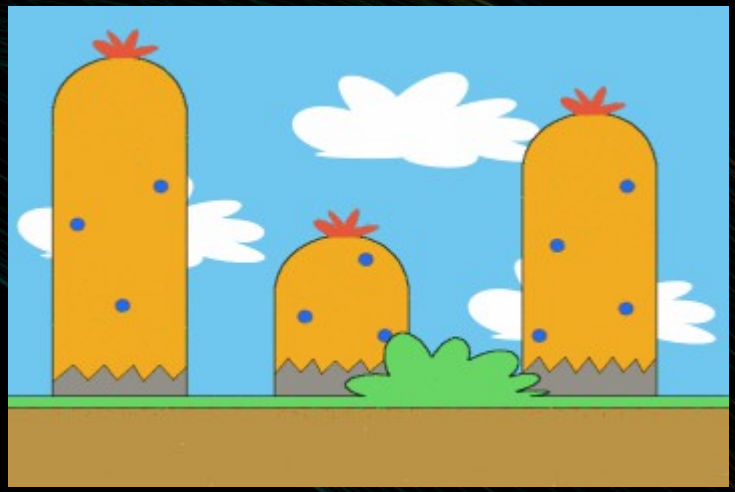
Movimento x Percepção

- Parallax:
  - Variação na percepção do movimento de acordo com a distância do objeto à lente;
  - Mesma tela, mesma cena, gerar objetos de perspectiva;
  - Tamanho de objeto e modo como se movimenta;

Movimento x Percepção

- Parallax:





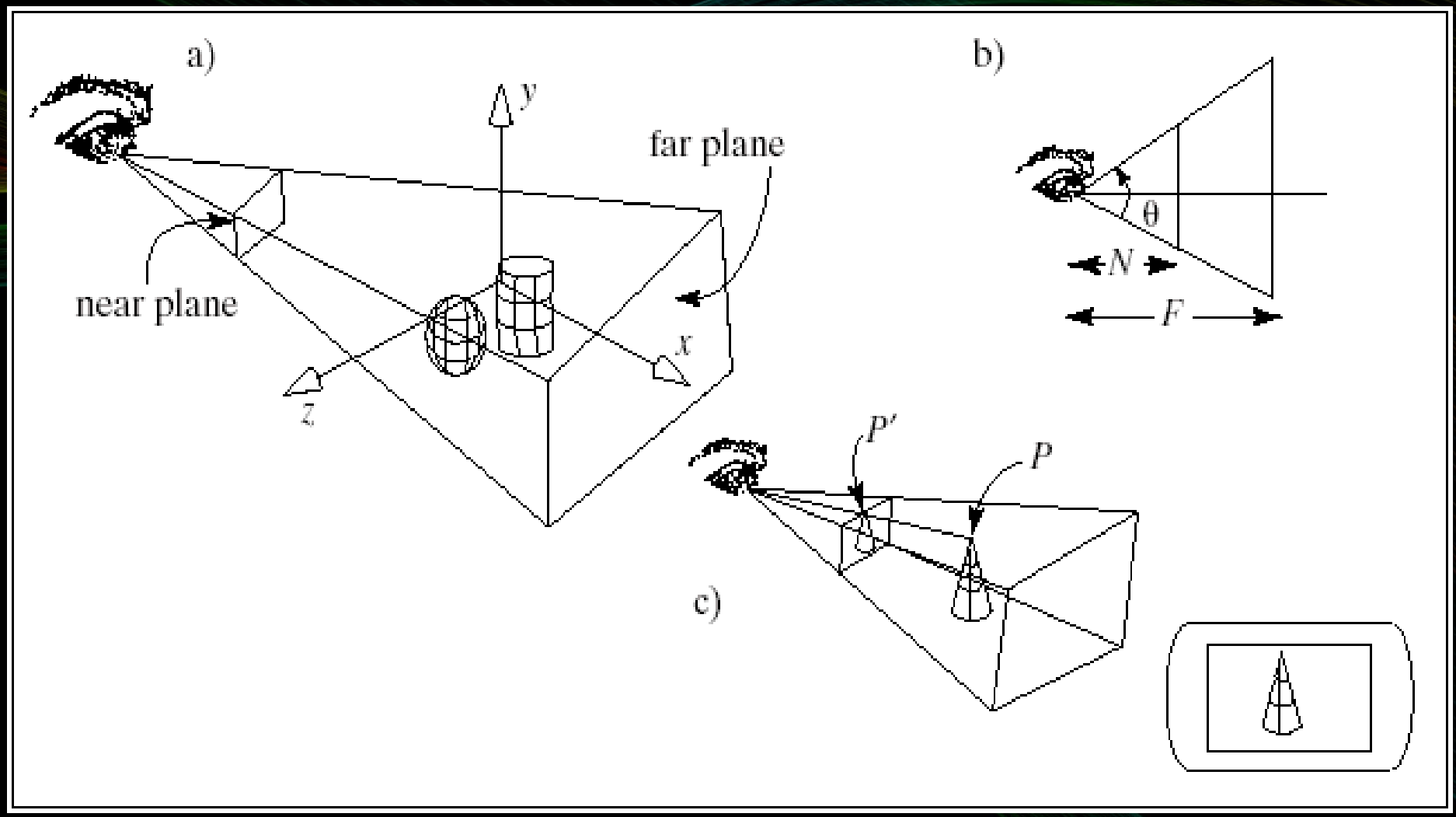
Introdução

- Necessário construir um ambiente de controle de câmera;
  - Facilidade de navegação entre objetos;
  - Transformações de câmera não afetam o objeto;

Câmera em Perspectiva

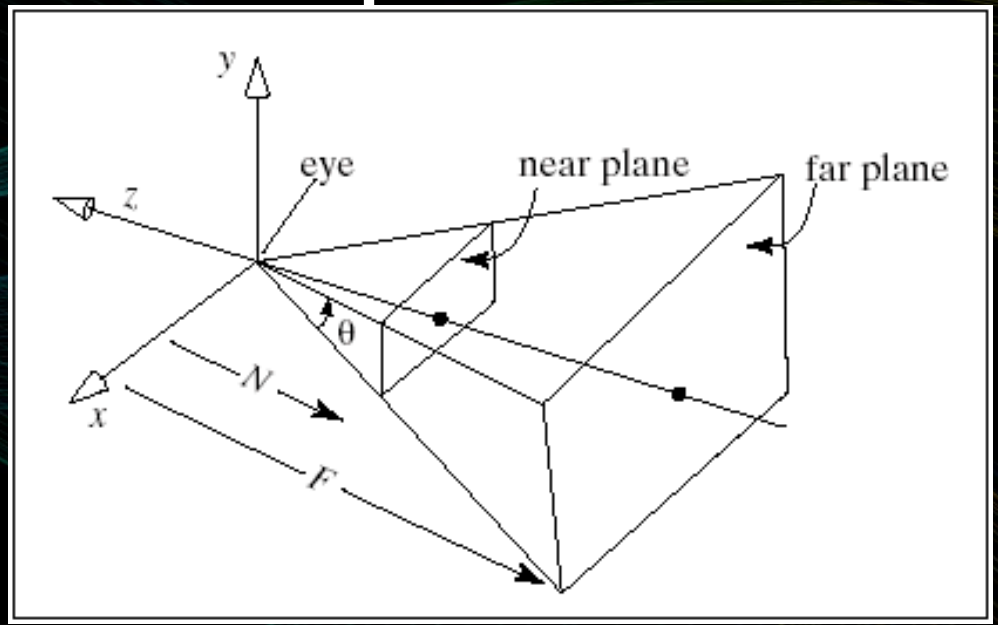
- Olho:
  - Posicionado em algum lugar na cena;
- Volume de visualização:
  - Parte entre o plano perto e longe;
- Ângulo de Visão (θ):
  - Abertura vertical da pirâmide de visualização;
- Plano de Visualização:
  - Plano visível entre perto e longe;
- Razão de Aspecto:
  - Resolução do plano de visualização (altura/largura)

Câmera em Perspectiva



Câmera em Perspectiva

Altura = *2*N*tg(θ/2)*
Largura = *Altura*razão de aspecto*



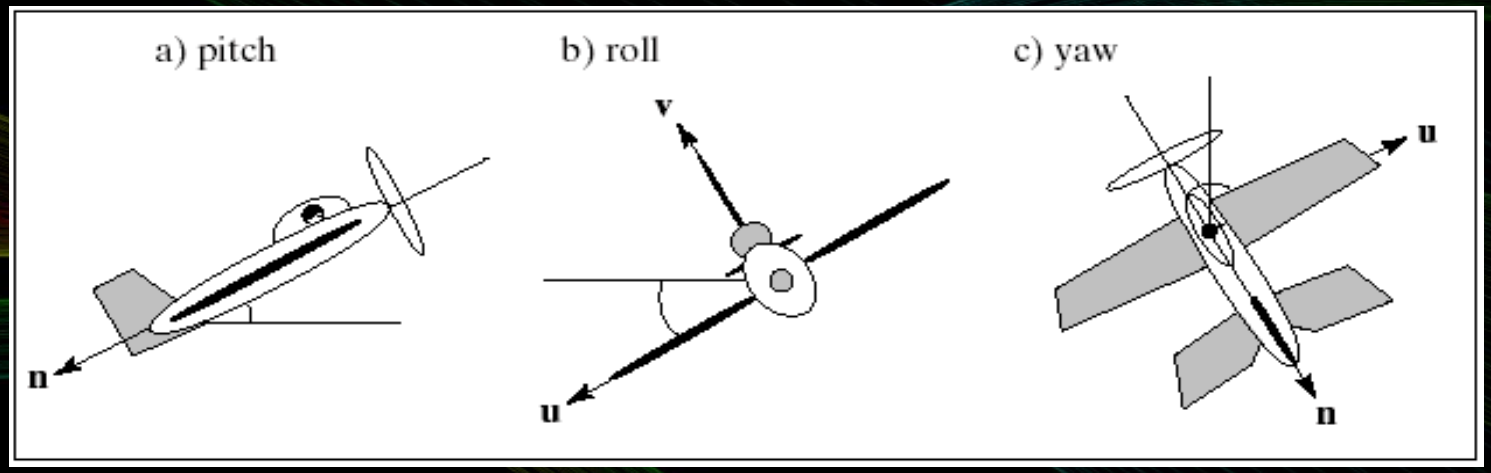
Criando o modelo de câmera:

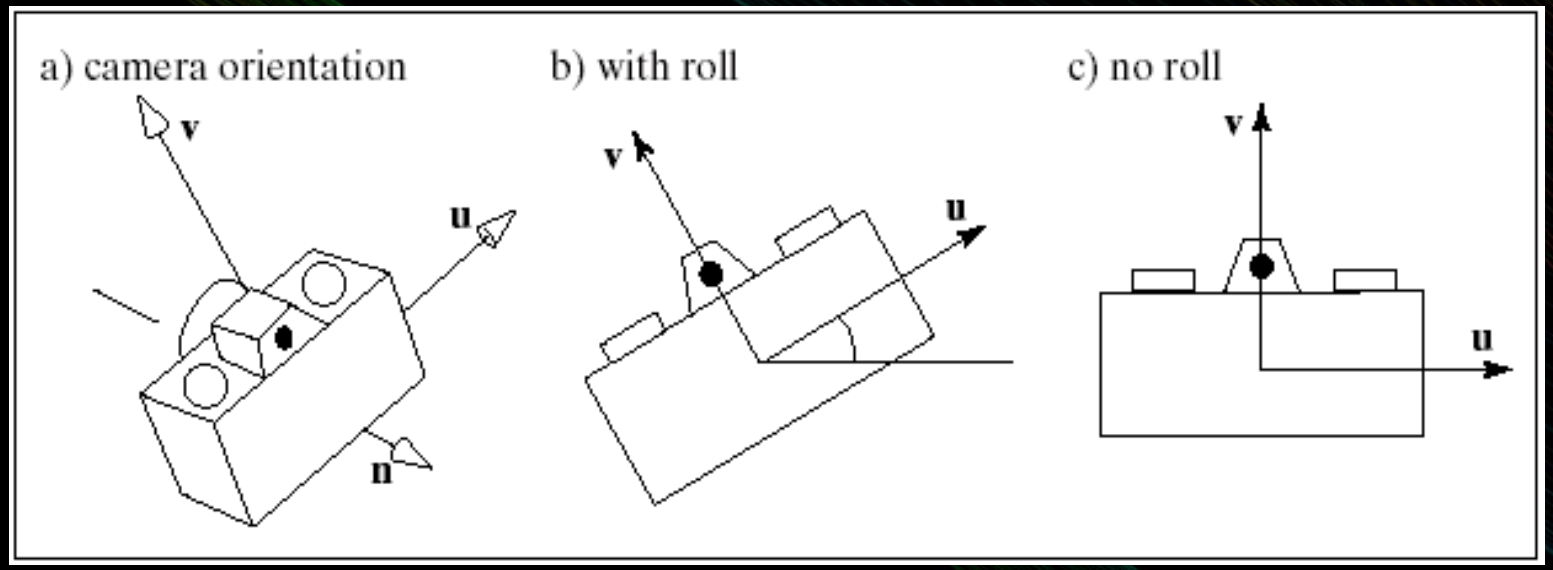
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(θ, largura/altura, N, F);
```

Posicionando a câmera:

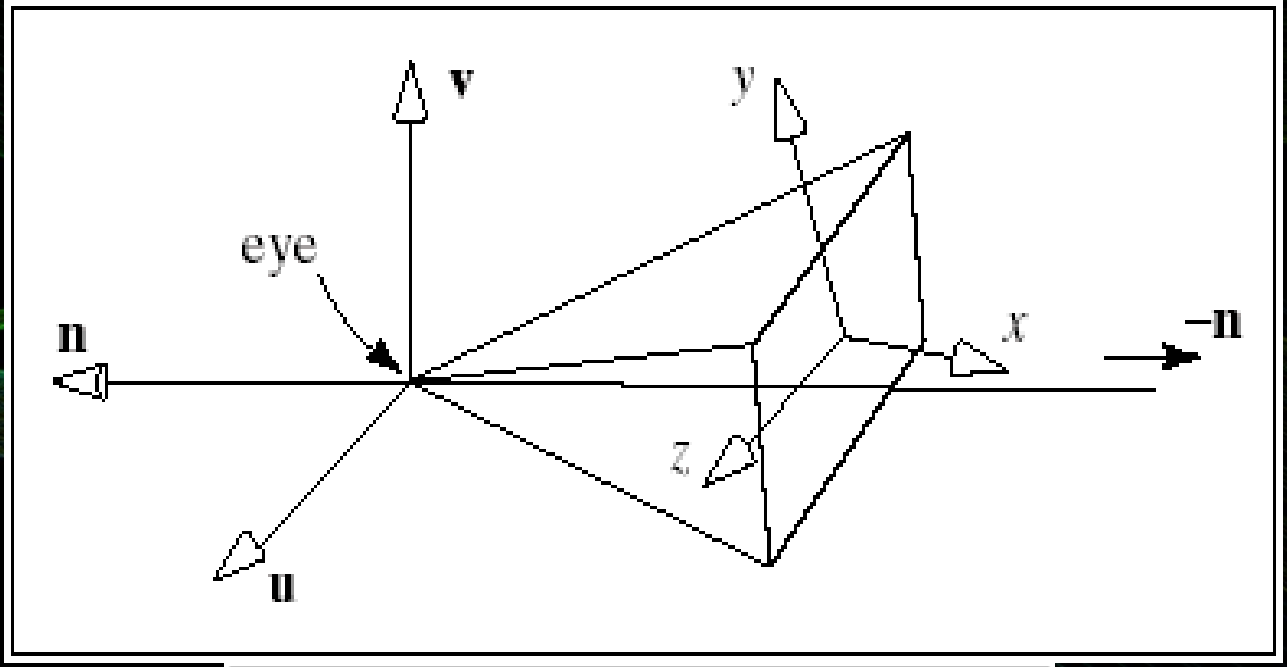
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z);
```

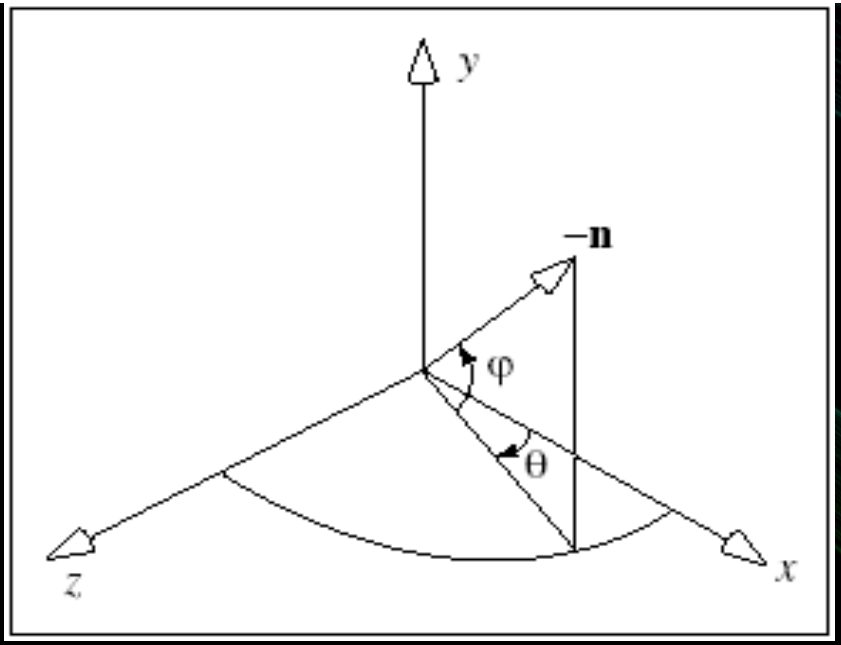
Movimentos de Câmera





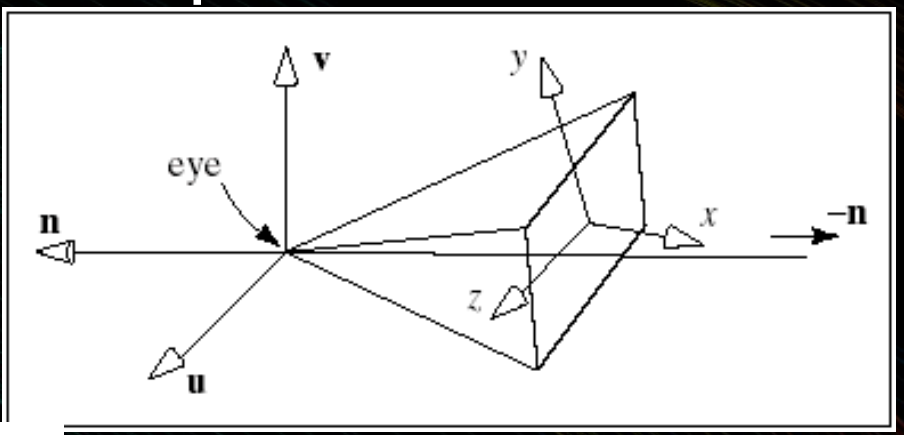
Movimentos de Câmera





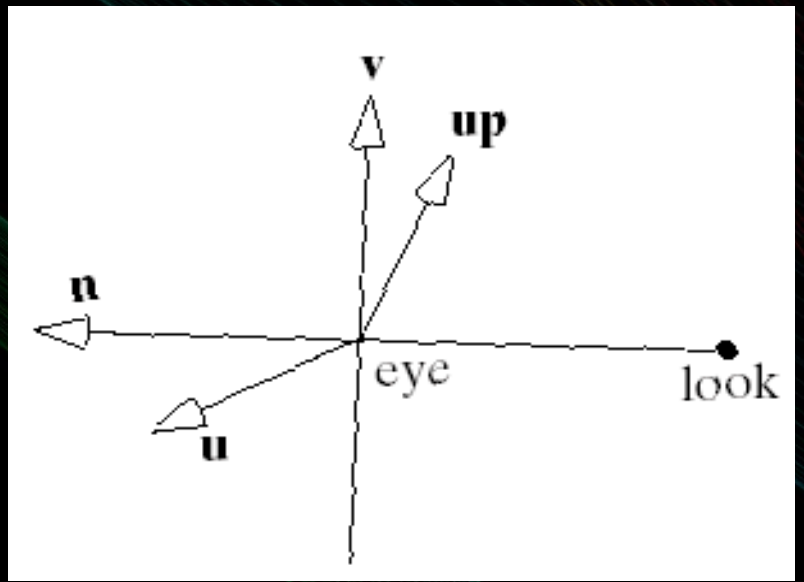
Direcionamento de Câmera

- A partir do vetores eye, look e up:
  - Temos:
    - n = eye – look;
    - u = up x n;
    - v = n x u;



$$v = \begin{pmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$onde \begin{pmatrix} d_x & d_y & d_z \end{pmatrix} = \begin{pmatrix} -eye.u & -eye.v & -eye.n \end{pmatrix}$$



Camera

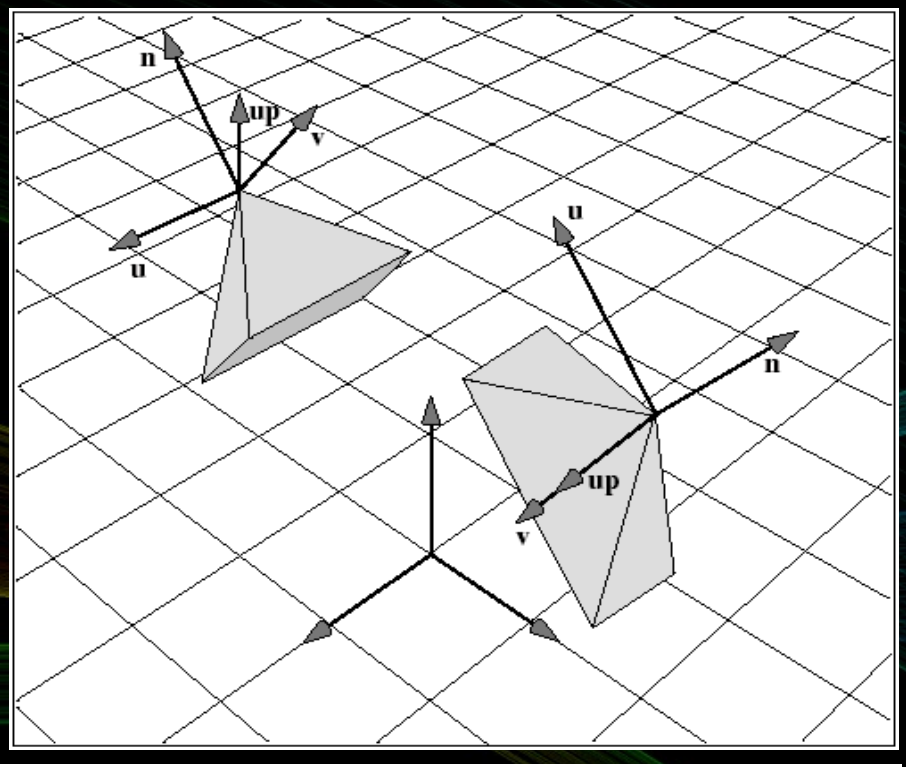
```
class Camera{
  private:
        Point3 eye;
        Vector3 u,v,n;
        double viewAngle, aspect, nearDist, farDist; // view volume shape
        void setModelViewMatrix(); // tell OpenGL where the camera is

  public:
        Camera(void){}; // default constructor
        void set(Point3 eye, Point3 look, Vector3 up); // like gluLookAt()
        void roll(float angle); // roll it
        void pitch(float angle); // increase pitch
        void yaw(float angle); // yaw it
        void slide(float delU, float delV, float delN); // slide it
    void rotate (Vector3 axis, float angle);
        void setShape(float vAng, float asp, float nearD, float farD);
};
```

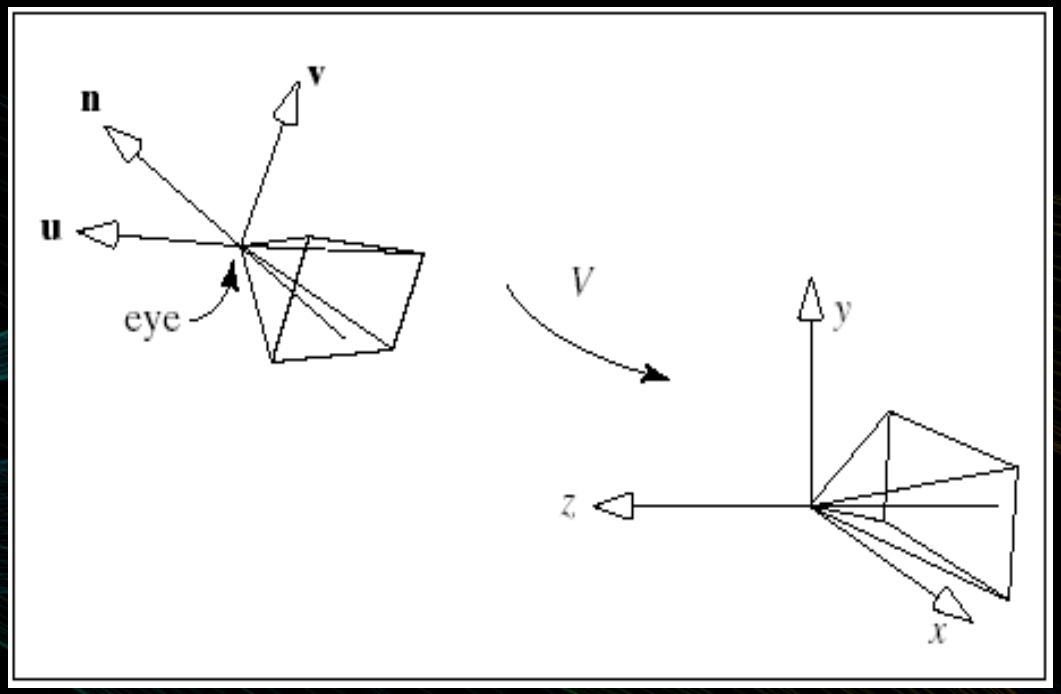
```
void Camera :: setShape(float vAngle, float asp, float nr, float fr){
    viewAngle = vAngle;
    aspect = asp;
    nearDist = nr;
    farDist = fr;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(viewAngle, aspect, nearDist, farDist);
    glMatrixMode(GL_MODELVIEW);
}

void Camera :: setModelViewMatrix(void){
// load modelview matrix with existing camera values
            float m[16];
            Vector3 eVec(eye.x, eye.y, eye.z); // a vector version of eye
            m[0] =  u.x; m[4] =  u.y; m[8]  =  u.z;  m[12] = -eVec.dot(u);
            m[1] =  v.x; m[5] =  v.y; m[9]  =  v.z;  m[13] = -eVec.dot(v);
            m[2] =  n.x; m[6] =  n.y; m[10] =  n.z;  m[14] = -eVec.dot(n);
            m[3] =  0;   m[7] =  0;   m[11] =  0;    m[15] = 1.0;
            glMatrixMode(GL_MODELVIEW);
            glLoadMatrixf(m); // load OpenGL's modelview matrix
}
```

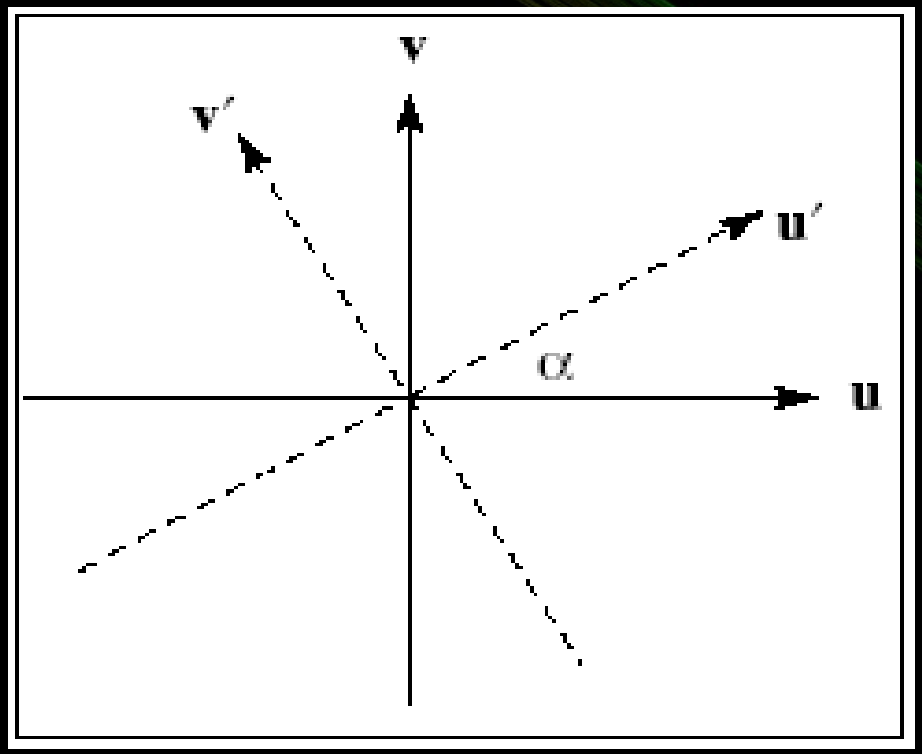
```
void Camera:: set(Point3 Eye, Point3 look, Vector3 up)
{       // create a modelview matrix and send it to OpenGL
        eye.set(Eye); // store the given eye position
    n.set(eye.x - look.x, eye.y - look.y, eye.z - look.z); // make n
        u.set(up.cross(n).x, up.cross(n).y, up.cross(n).z); // make u = up X n
        n.normalize(); u.normalize(); // make them unit length
        v.set(n.cross(u).x, n.cross(u).y, n.cross(u).z);  // make v =  n X u
        setModelViewMatrix(); // tell OpenGL
}

void Camera:: slide(float delU, float delV, float delN)
{
    eye.x += delU * u.x + delV * v.x + delN * n.x;
    eye.y += delU * u.y + delV * v.y + delN * n.y;
    eye.z += delU * u.z + delV * v.z + delN * n.z;
    setModelViewMatrix();
}
```

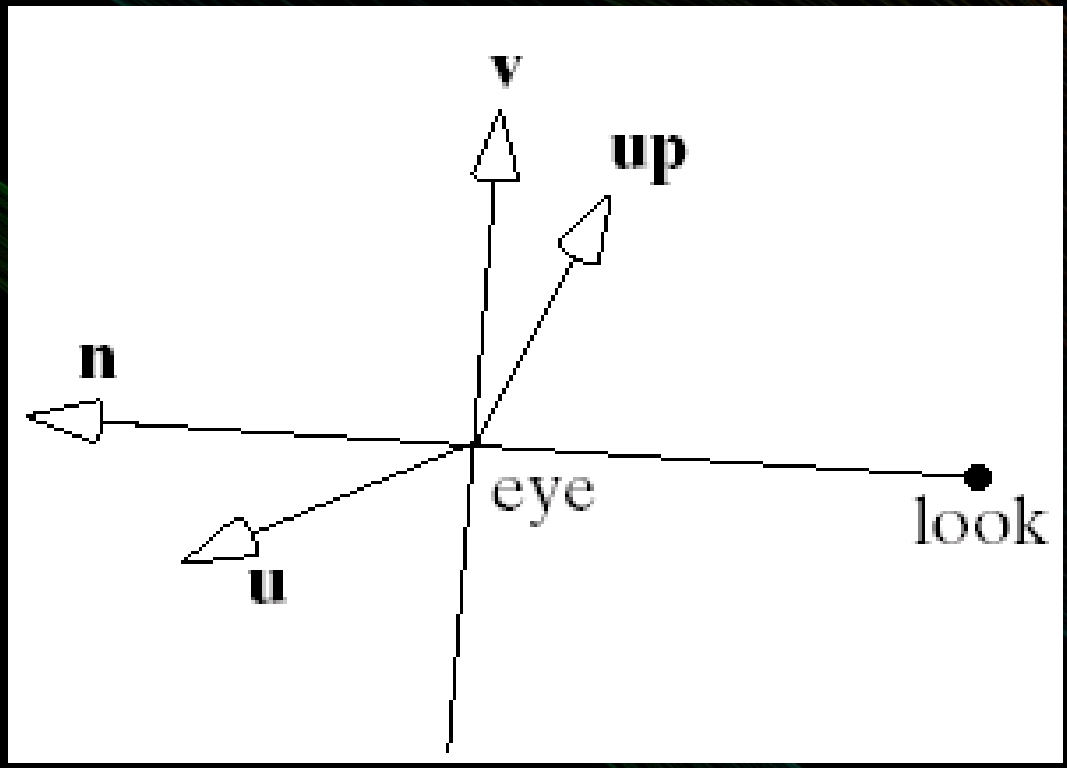
```
void Camera:: roll(float angle)
{//u´=cos(α)u+sin(α)v, v´=-sin(α)u+cos(α)v
    float cs=cos(PI/180.0 * angle);
    float sn=sin(PI/180.0 * angle);
    Vector3 t = u;
    u.set(cs*t.x - sn*v.x, cs*t.y - sn*v.y, cs*t.z - sn*v.z);
    v.set(sn*t.x + cs*v.x, sn*t.y + cs*v.y, sn*t.z + cs*v.z);
    setModelViewMatrix();
}

void Camera :: pitch (float angle)
{ // pitch the camera through angle degrees around U
    float cs = cos(3.14159265/180 * angle);
    float sn = sin(3.14159265/180 * angle);
    Vector3 t(v); // remember old v
    v.set(cs*t.x - sn*n.x, cs*t.y - sn*n.y, cs*t.z - sn*n.z);
            n.set(sn*t.x + cs*n.x, sn*t.y + cs*n.y, sn*t.z + cs*n.z);
            setModelViewMatrix();
}
```

```
void Camera :: yaw (float angle)
{ // yaw the camera through angle degrees around V
    float cs = cos(3.14159265/180 * angle);
    float sn = sin(3.14159265/180 * angle);
    Vector3 t(n); // remember old v
    n.set(cs*t.x - sn*u.x, cs*t.y - sn*u.y, cs*t.z - sn*u.z);
            u.set(sn*t.x + cs*u.x, sn*t.y + cs*u.y, sn*t.z + cs*u.z);
            setModelViewMatrix();
}
```


n
up
v
u
u
n
up
v


v
n
u
eye
V
y
z
x


v
v′
u′
α
u


v
up
n
eye
look
u

Texto Base

- Hill, F. S. Jr.; *Computer Graphics using OpenGL*, Pearson Education, 2a edição. 922p, 2001.

Atividade 07/1

- Adicionar a classe Camera na atividade 06/2;
- Adicionar as seguintes operações à câmera:
  - Z afasta a câmera, z aproxima;
  - P e p rotação anti-horária e horária em relação ao eixo x da câmera (Pitch);
  - R e r rotação anti-horária e horária em relação ao eixo z da câmera (Roll);
  - Y e y rotação anti-horária e horária em relação ao eixo y da câmera (Yaw);
- À partir deste exercício, podem usar as funções de translação, escala e rotação do OpenGL;

Dúvidas?

Obrigado.