Projeção e Visão Tridimensional

José Luis Seixas Junior

Ciência da Computação
Universidade Estadual do Paraná

Computação Gráfica
2017

Índice

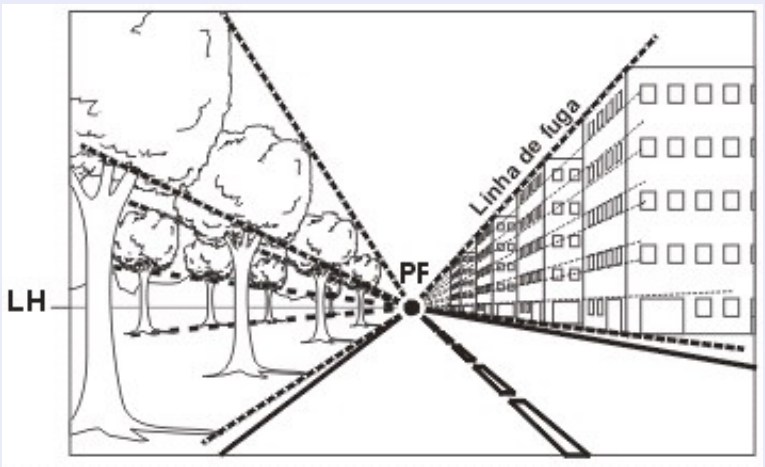
1. Introdução
2. Projeção em Plano de Imagem
3. Movimento e Percepção
4. Camera
5. Atividade

Introdução

Realismo

- Criar realismo;
- Projeção em perspectiva;
- Linha de fuga;

Centro de projeção



Introdução

Exemplo

Introdução

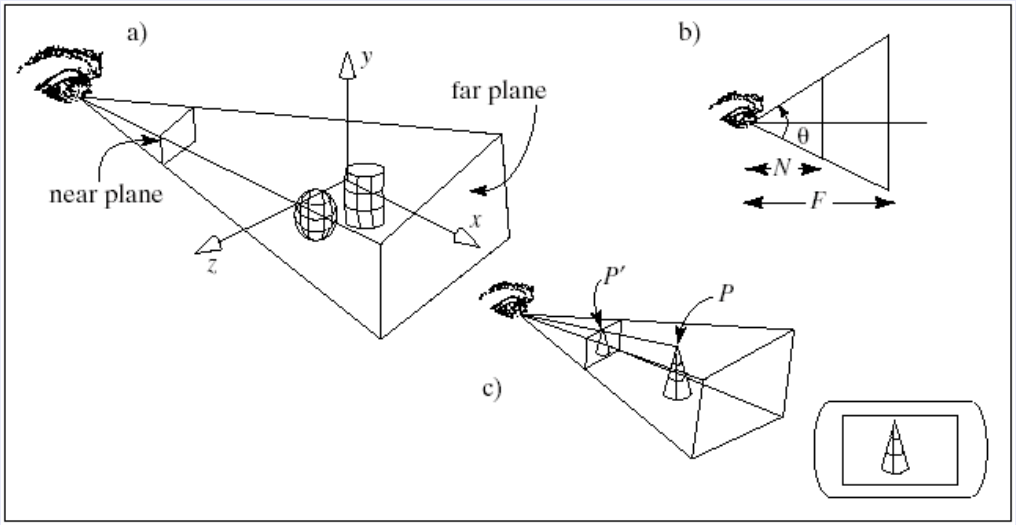
Efeitos

Introdução

Controle de Câmera

- Facilita a navegação entre objetos;
- Transformações não afetam o objeto;

Centro de projeção

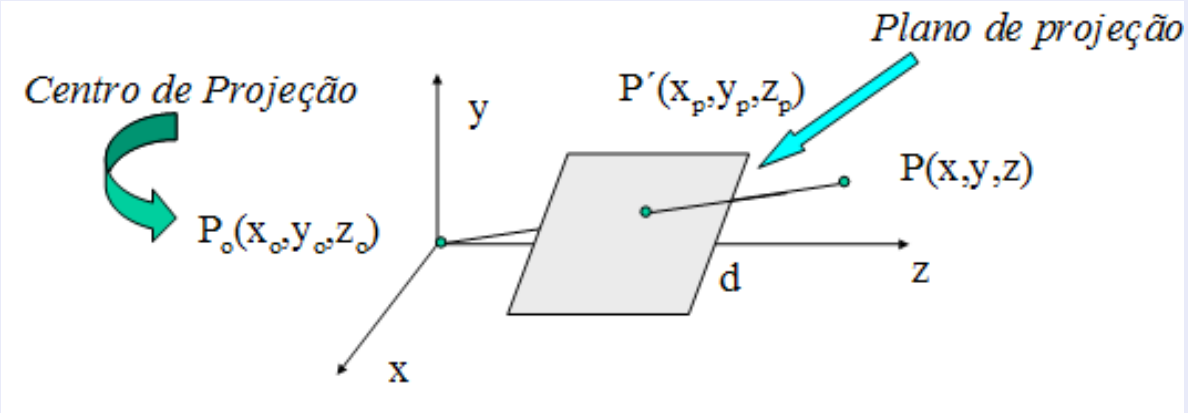


Projeções Geométricas

Planares

- Plano de Projeção;
- Centro de Projeção;

Centro de projeção

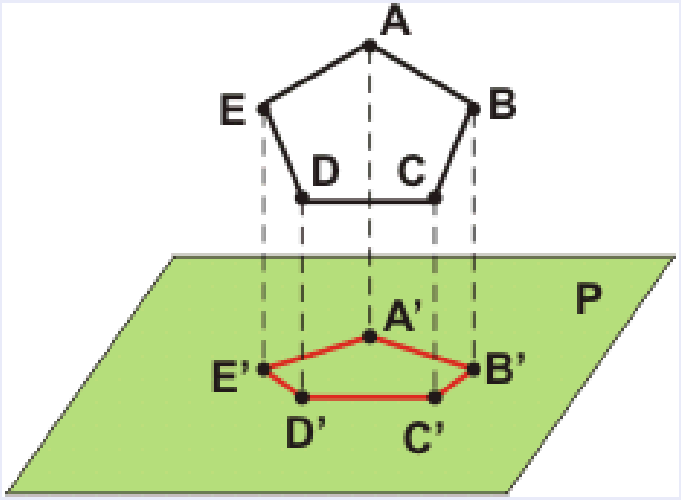


Projeções Geométricas

Distância no Infinito

- Centro de projeção infinitamente afastado;
- Projeção Paralela;

Centro de projeção

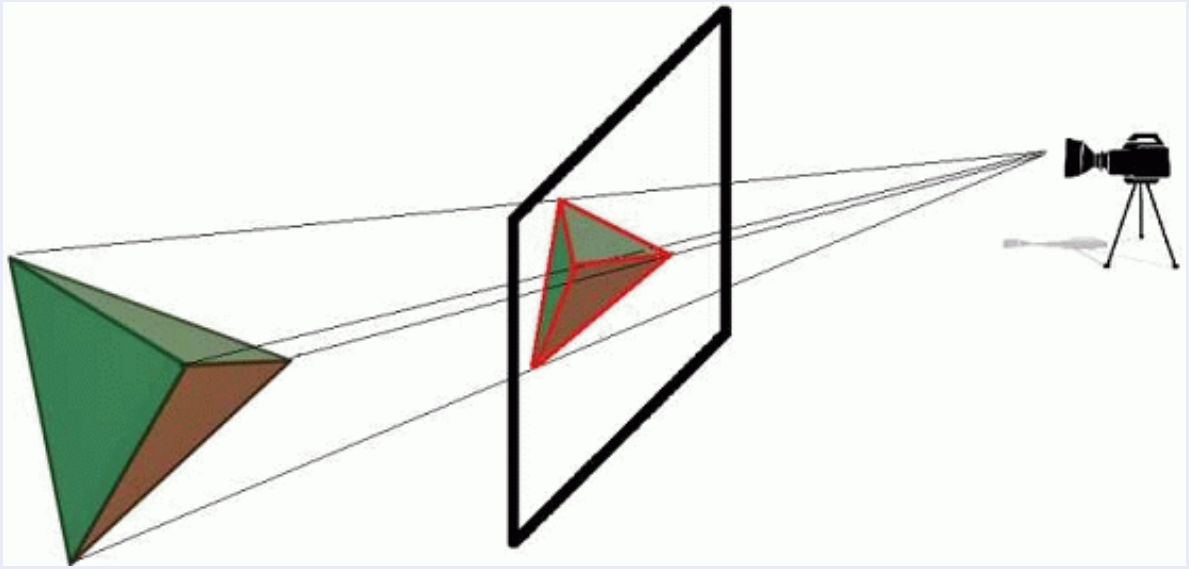


Projeções Geométricas

Planares

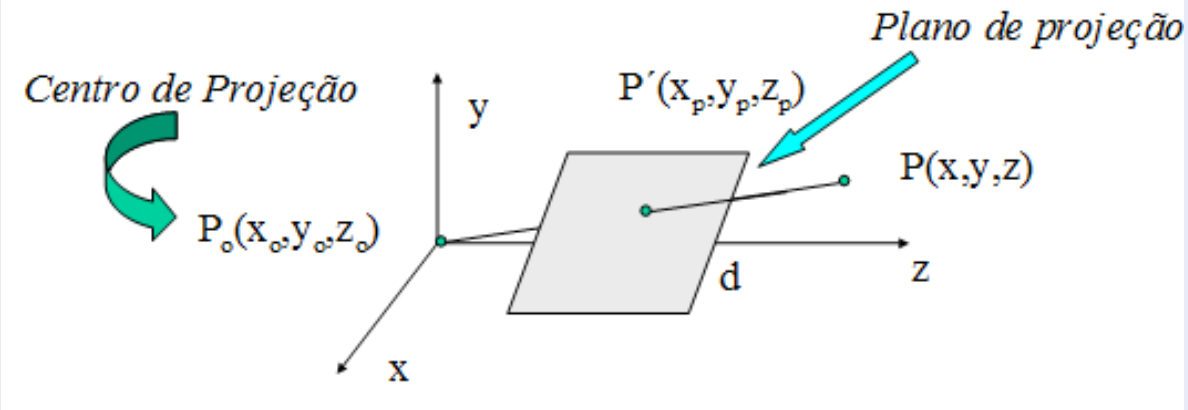
- Centro em distância finita;
- Projeção em Perspectiva;

Centro de projeção



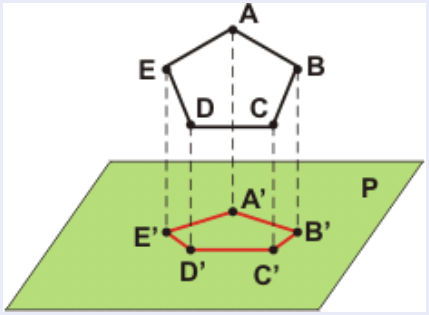
Projeções Geométricas

Centro de projeção



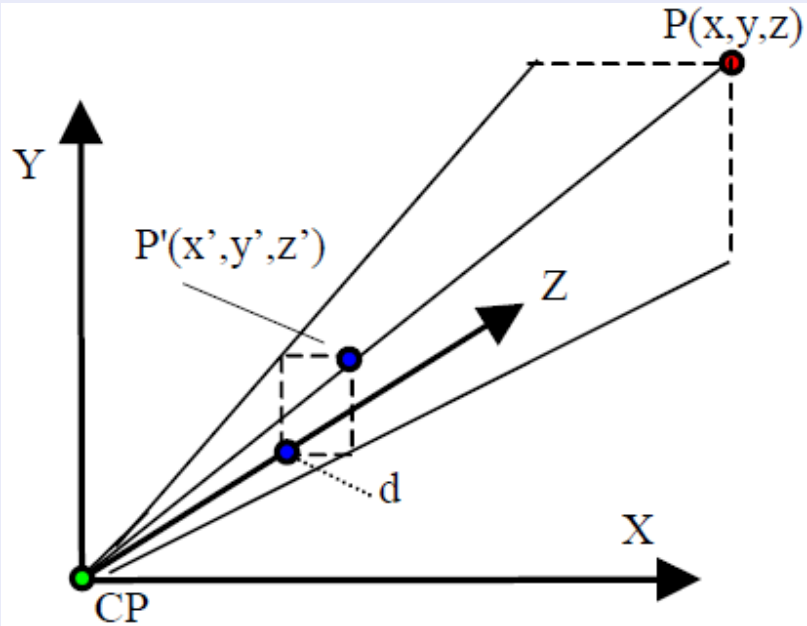
Matematicamente

$$\frac{x_p}{z_p} = \frac{x}{z}, \qquad \frac{y_p}{z_p} = \frac{y}{z}$$



Projeções Geométricas

Distância de centro



Matematicamente

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{X}{W} \\ \frac{Y}{W} \\ \frac{Z}{W} \\ 1 \end{bmatrix} = \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

Parallax

Percepção

- Distância do objeto para a lente;
- Mesma tela, mesma cena;
- Tamanho de objeto ou modo como se movimenta;

Parallax

Exemplo

Camera em Perspectiva

Olho

- Posicionameno sobre um lugar na cena;

Volume de visualização

- Volume da figura gerada entre os planos de perto e longe;

Ângulo de visão (θ)

- Abertura vertical da pirâmide de visualização

Camera em Perspectiva

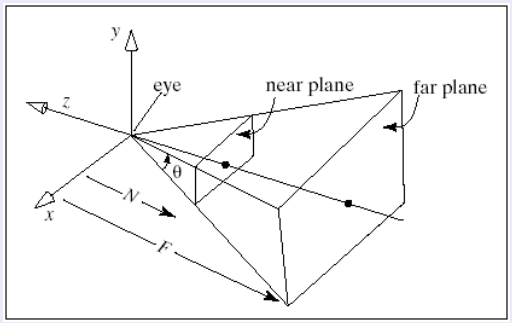
Plano de Visualização

- Projeção visível entre planos perto e longe;

Razão de Aspecto

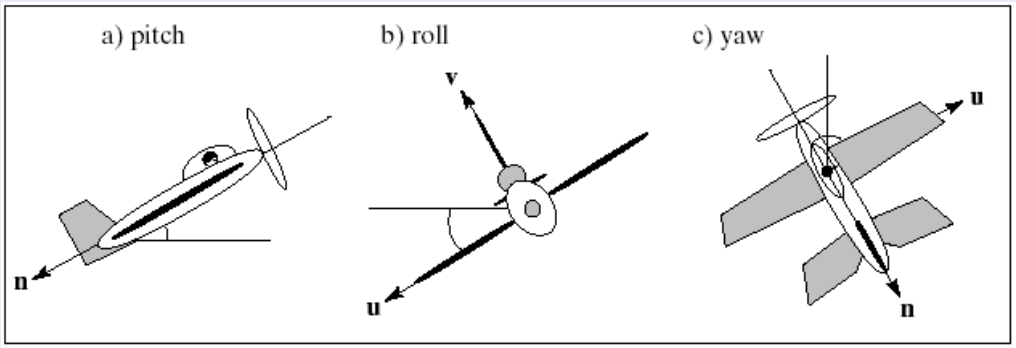
- Resolução do plano;
- Altura/Largura do plano;

Imagem

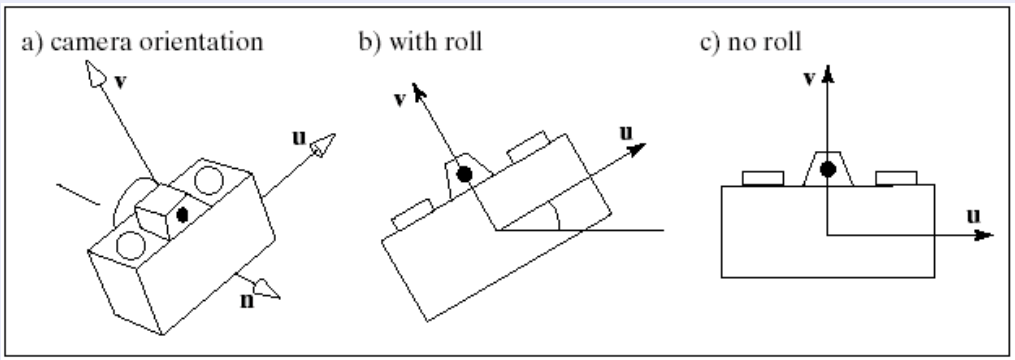


Movimentos de Câmera

Movimentos



Câmera



Movimentos de Câmera

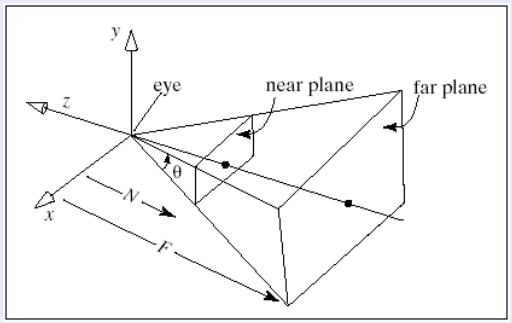
Movimentos

Câmera



Camera em Perspectiva

Código



Criando o modelo de câmera:

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

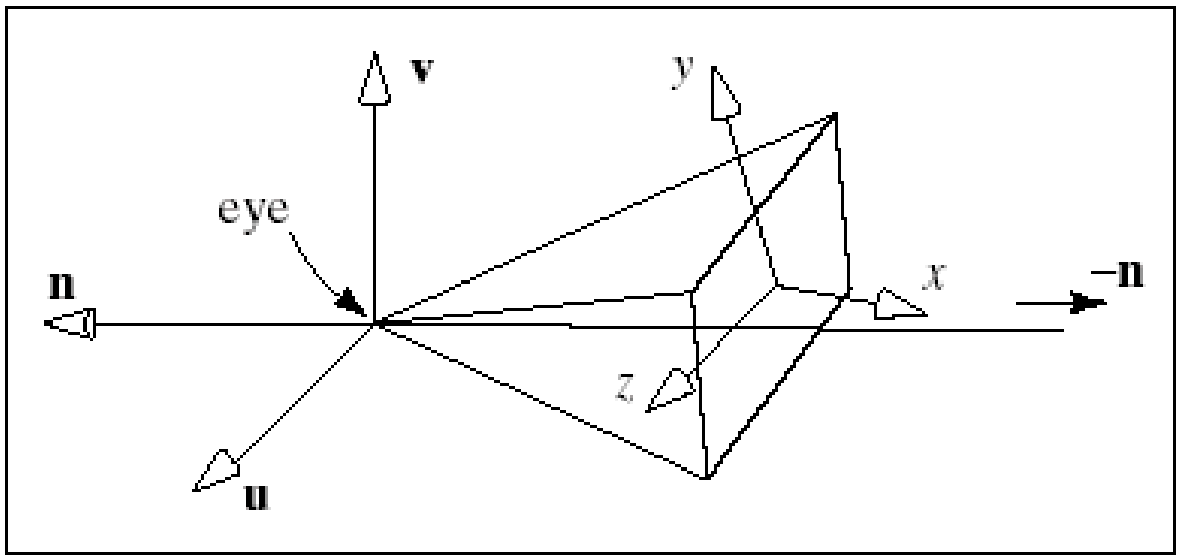
Posicionando a câmera:

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

Componentes de Câmera

Componentes

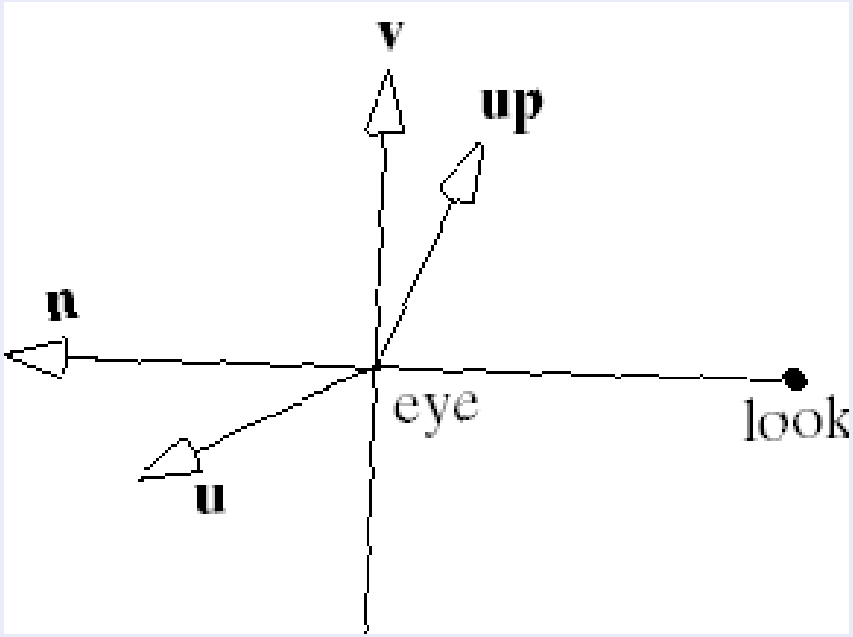
- eye → posição do olho;
- look → apontamento de onde esta olhando;
- up → apontamento de onde é pra cima;



Componentes de Câmera

A partir das Componentes

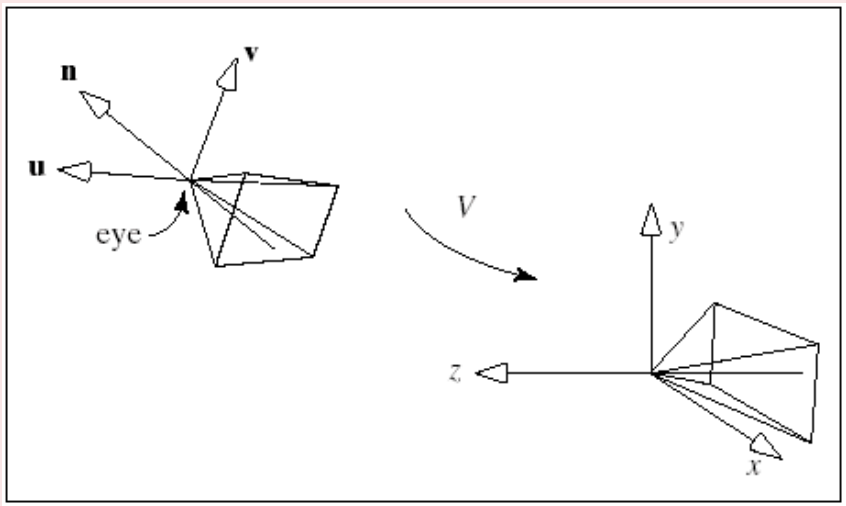
- $n = eye - look$
- $u = upxn$
- $v = nxu$



Componentes de Câmera

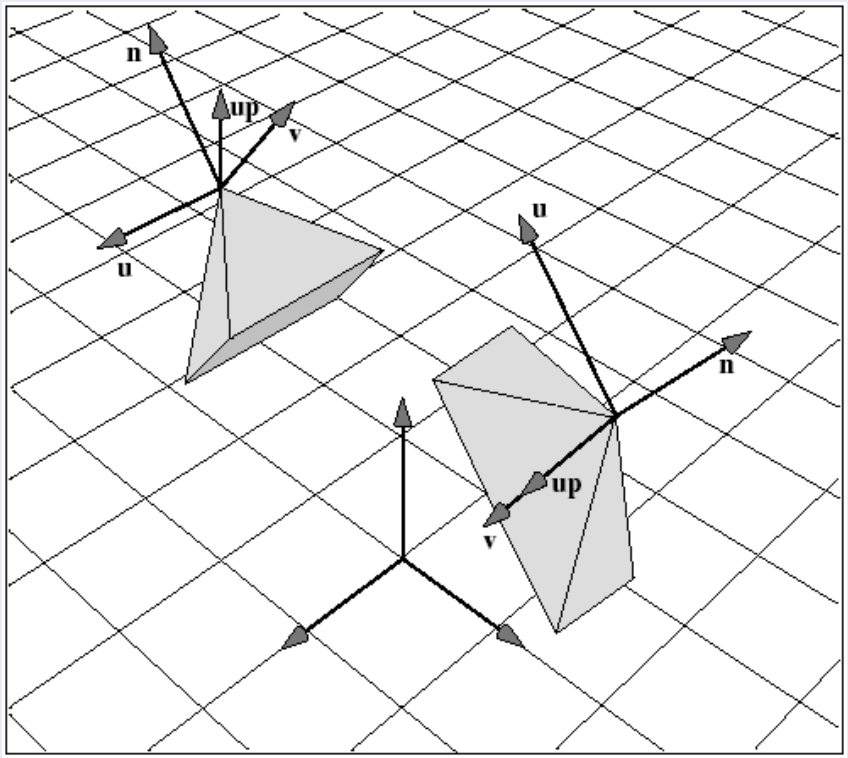
Camera $\neq$ Mundo

- $x \neq n$
- $y \neq v$
- $z \neq u$



Componentes de Câmera

Coordenadas e Multicâmeras



A Classe Camera

Camera.h

```
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);
};
```

A Classe Camera

setShape

```
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);
}
```

A Classe Camera

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
}
```

A Classe Camera

set

```
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
}
```

A Classe Camera

slide

```
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();
}
```

A Classe Camera

roll

```
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();
}
```

A Classe Camera

pitch

```
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();
}
```

A Classe Camera

yaw

```
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();
}
```

Atividade 08

Atividade 08/1

- Adicionar a classe de Camera na atividade 07/1:
- Operações de câmera:
  - 'Z' afasta, 'z' aproxima a câmera;
  - 'P' pitch horário, 'p' anti-horário;
  - 'R' roll horário, 'r' anti-horário;
  - 'Y' yaw horário, 'y' anti-horário;

Data

09 de novembro de 2017

Atividade 08

Observação

A partir deste exercício, é permitido o uso de funções translação, escala e rotação do OpenGL.

Referências I


Hill, F. S.
Computer Graphics Using OpenGL.
Prentice Hall, 2013.


Shreiner, D.; Woo M.; Neider, J.; Davis, T.
OpenGL Programming Guide.
Addison Wesley, 4º edição, 2013.