Transformaciones

Néstor Gómez
ngomez@itesm.mx

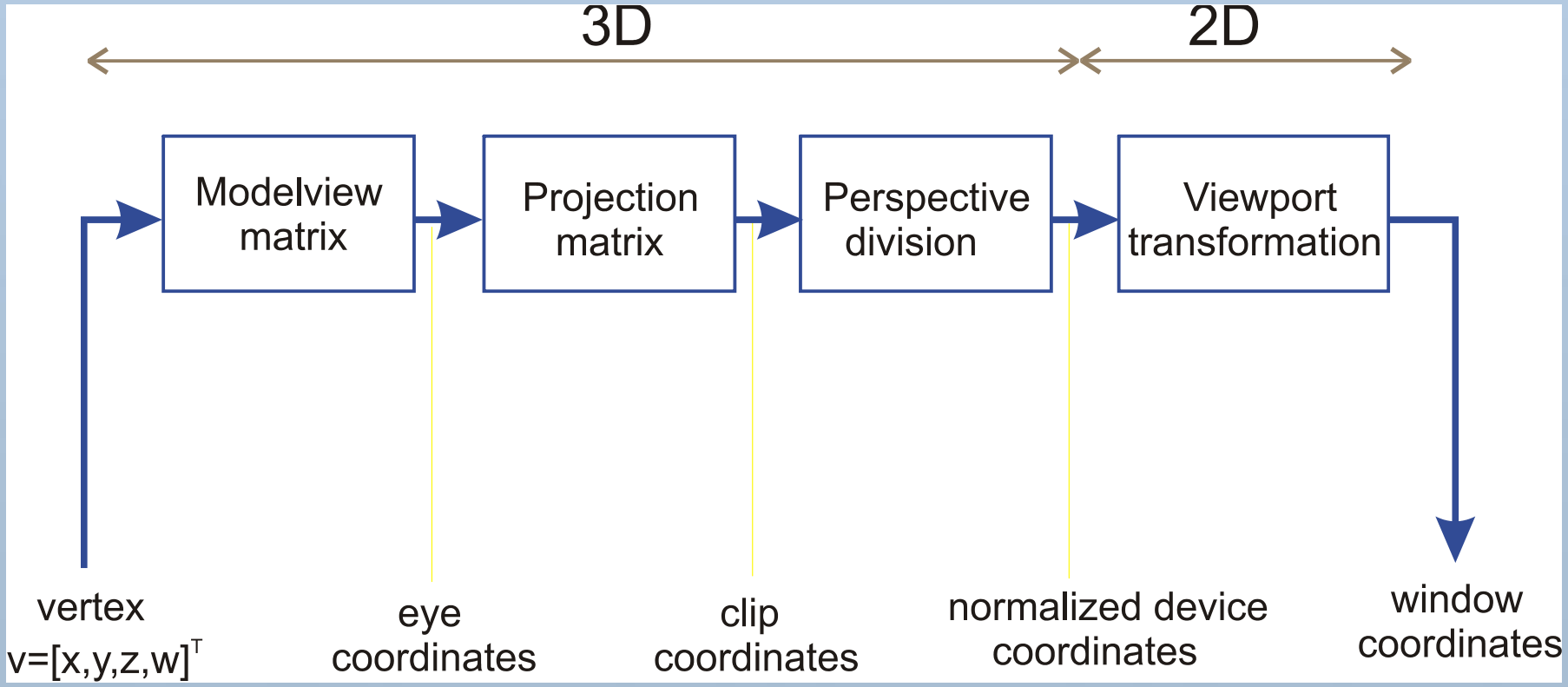
Transformaciones de Coordenadas

TECNOLÓGICO DE MONTERREY.

- Los objetos son usualmente modelados en la posición inicial (en el origen)
- Una forma de llevar un objeto a una diferente posición es transformándolo
- Las transformaciones de un vértice pueden ser:

  - Viewing
  - Modeling
  - Projection
  - Clipping
  - Viewport

Transformaciones de Coordenadas



Transformaciones de Coordenadas

TECNOLÓGICO DE MONTERREY.

Etapas de las transformaciones de un vértice

- La etapa de viewing consiste de dos partes:
  - Posicionamiento y proyección
- OpenGL soporta proyecciones perspectiva y ortográfica
- Otras proyecciones pueden definirse a mano
- El sistema de coordenadas es representado por una matriz
- Las transformaciones son representadas por una multiplicación de matrices
- La matriz usada es de 4x4

TECNOLÓGICO DE MONTERREY.

Matrices

- Las coordenadas del vértice transformado v' son calculadas de:
  - Coordenadas v
  - Matriz M
  - v' = Mv (siempre en este orden)

$$M = \begin{bmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{bmatrix}$$

Modelview, Projection, Texture

Podemos hacer las siguientes operaciones

- Transformar objetos
- Transformar la cámara
- Transformar coordenadas de textura

Funciona de la siguiente manera:

- Establecer las transformaciones
- Enviar datos (los datos serán transformados)
- PRIMERO debemos definir la transformación y DESPUÉS se envían los datos para que se transformen

Transformaciones de Coordenadas

- Las transformaciones de viewing y modeling son combinadas en una sola matriz

- Hay dos maneras de establecer las transformaciones de la cámara
  - Mover los objetos (modeling transformation)
  - Mover la cámara (viewing transformation)

Modelview, Projection, Texture

```
void glMatrixMode(GLenum mode)
```

Donde

mode es **GL_MODELVIEW**

- o **GL_PROJECTION**
- o **GL_TEXTURE**

Las funciones subsecuentes afectarán a la matriz especificada

Modelview, Projection, Texture

TECNOLÓGICO DE MONTERREY.

```
void glLoadIdentity(void)
```

Establece la matriz identidad en la matriz actual

```
void glTranslate{fd}(TYPE x,TYPE y,TYPE z)
```

Multiplica la matriz actual por la matriz de traslación (x,y,z)

```
void glScale{fd}(TYPE x,TYPE y,TYPE z)
```

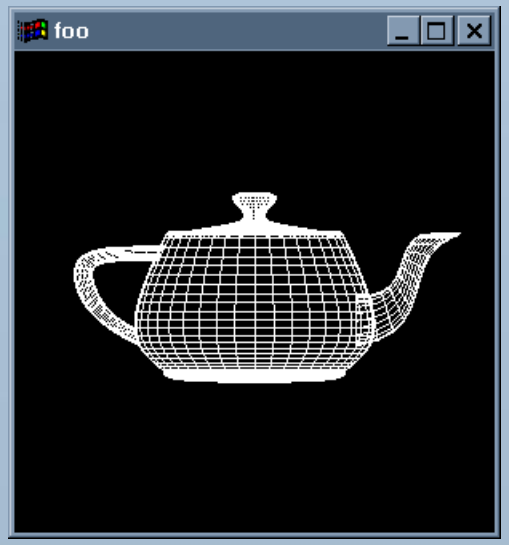
Multiplica la matriz actual por la matriz de escalado (x,y,z)

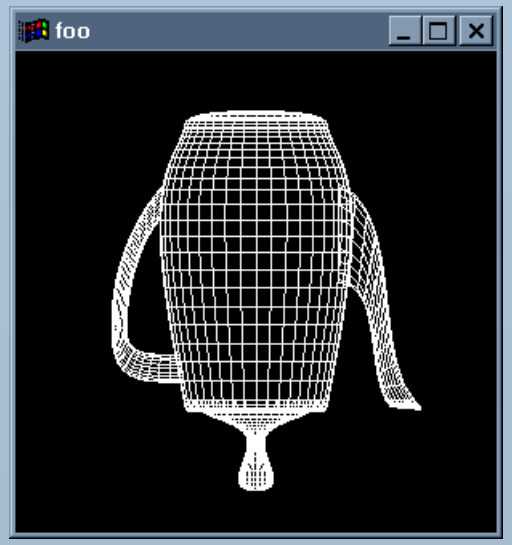
Transformaciones

TECNOLÓGICO DE MONTERREY.

Ejemplo:

```
glLoadIdentity();
DrawObject();
glScalef(0.8, -2.0, 1.3);
DrawObject();
```





Transformaciones

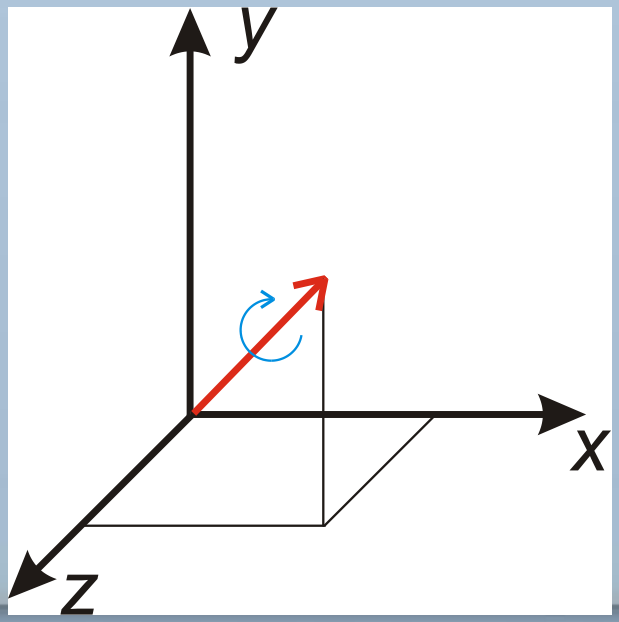
```
void glRotate{fd}(TYPE angle,
                  TYPE x,TYPE y,TYPE z)
```

Donde :

angle especifica el ángulo de rotación

x,y,z especifíca el ángulo de rotación

El ángulo es en grados [0-360]

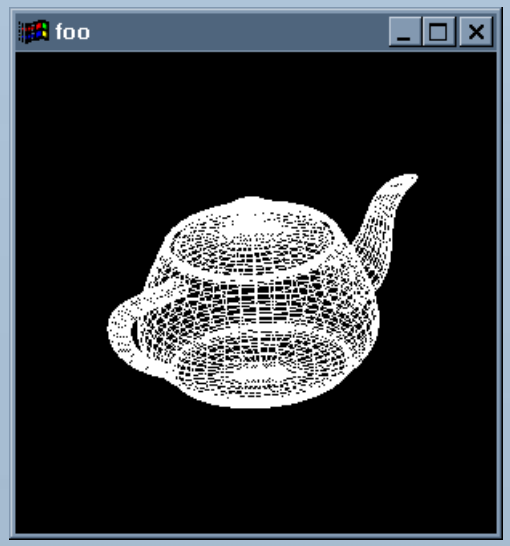


Transformaciones

TECNOLÓGICO DE MONTERREY.

Ejemplo:

```
glLoadIdentity();
DrawObject();
glRotatef(45,-1,-1.5,0.5);
DrawObject();
```



Transformaciones

TECNOLÓGICO DE MONTERREY.

```
void glLoadMatrix{fd}(const TYPE *m)
```

Establece los valores de la matriz activa por los valores especificados en **m**

```
glMatrixMode(GL_PROJECTION);
glLoadMatrix(myMatrix);
```

Es la manera de definir tu propia proyección

Transformaciones


```
void glMultMatrix{fd}(const TYPE *m)
```

Multiplica el valor de la matriz actual por los 16 valores especificados en **m** y guarda el resultado en la matriz actual

La matrices son multiplicadas de la siguiente manera

$$C \leftarrow C\ M$$

c - matriz actual, m – nueva matriz

Transformaciones

```
void glGetFloatv(GLenum matrix, GLfloat *m)
```

Donde:

matrix es **GL_MODELVIEW_MATRIX**

o **GL_PROJECTION_MATRIX**

o **GL_TEXTURE_MATRIX**

m es un apuntador a una matriz 4x4

Los valores de la matriz especificada se regresa en **m**

Transformaciones

Las matrices que enviamos y recibimos las debemos definir como `GLdouble m[16];`

```
GLdouble m[16] = {m0,m1,m2,m3,m4,m5,m6,m7,m8,
                  m9,m10,m11,m12,m13,m14,m15};
```

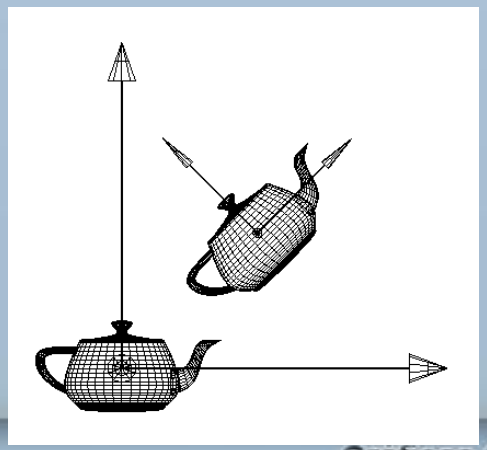
$$M = \begin{bmatrix} m0 & m4 & m8 & m12 \\ m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \end{bmatrix}$$

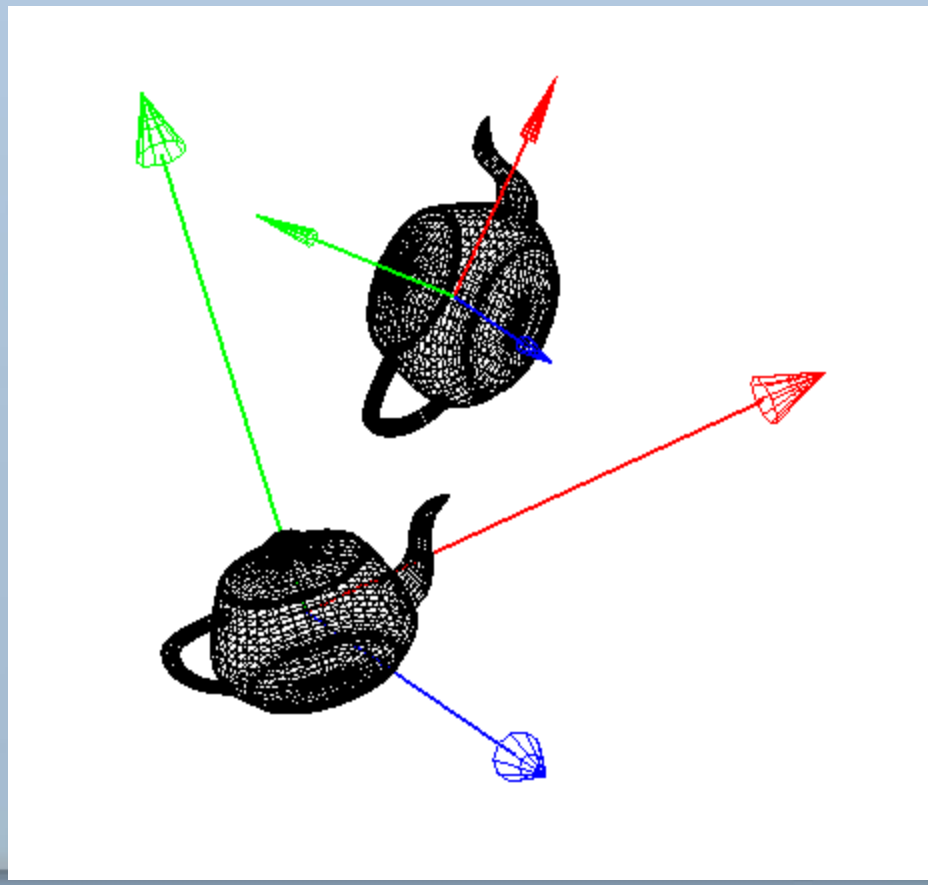
Transformaciones

TECNOLÓGICO DE MONTERREY.

El orden de las transformaciones es crítico

```
glMatrixMode(GL_MODELVIEW);
glColor3ub(0,0,0);
glutWireTeapot(0.3);
glRotatef(45,0,0,1);
glTranslatef(1,0,0);
glutWireTeapot(0.3);
```



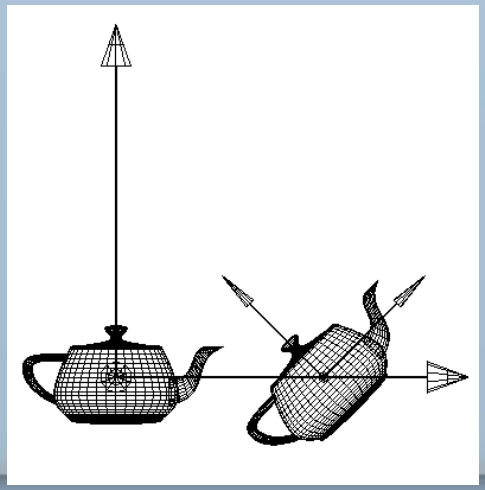


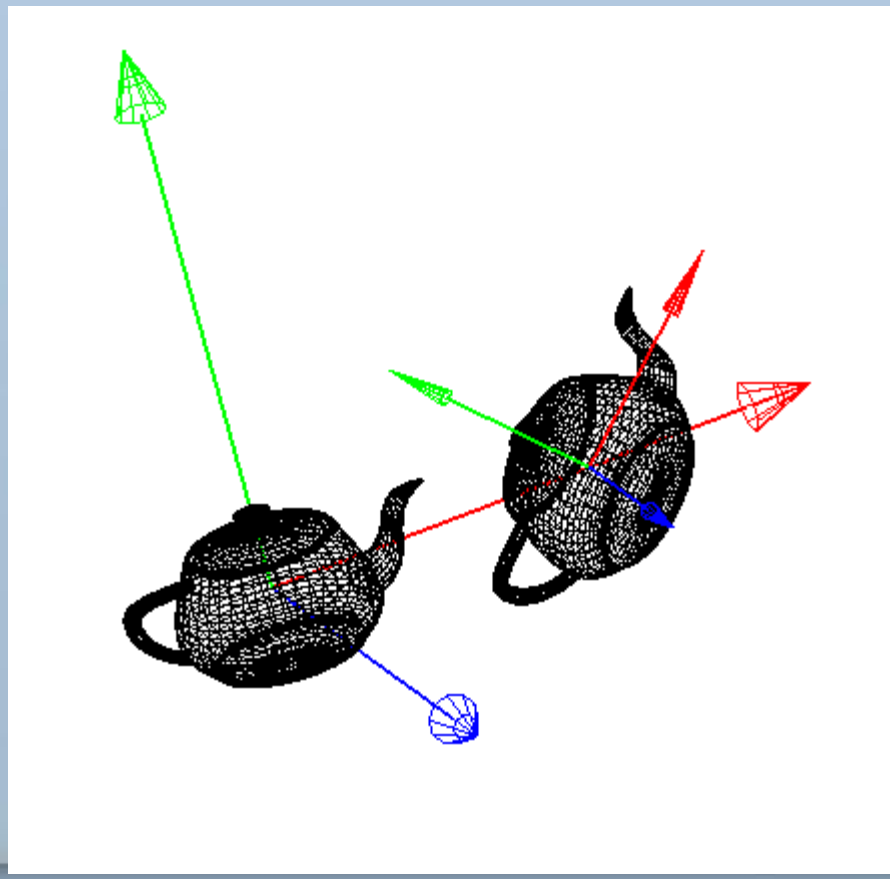
Transformaciones

TECNOLÓGICO DE MONTERREY.

El orden de las transformaciones es crítico

```
glMatrixMode(GL_MODELVIEW);
glColor3ub(0,0,0);
glutWireTeapot(0.3);
glTranslatef(1,0,0);
glRotatef(45,0,0,1);
glutWireTeapot(0.3);
```





Transformaciones

El orden de las transformaciones es crítico por que el orden de la multiplicación de matrices no es conmutativa

C´=RT

C”=TR

C’!=C”

Transformaciones

La primer alternativa:

El gran sistema de coordenadas

Debe ser leído en el orden inverso. Pensando en la composicón de matrices

La segunda alternativa:

El sistema de coordenadas locales atado al objeto

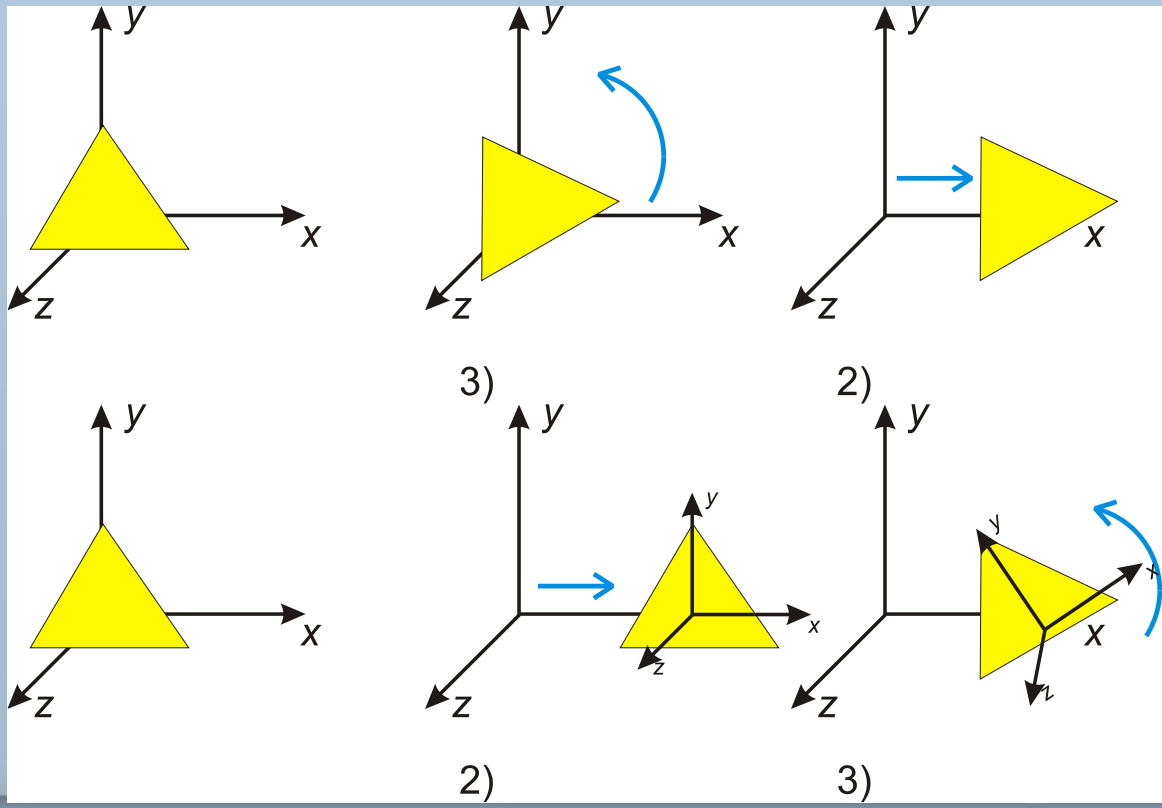
Las operaciones son aplicadas al sistema de coordenadas locales, las operaciones ocurren en el orden natural

Transformaciones

TECNOLÓGICO DE MONTERREY.

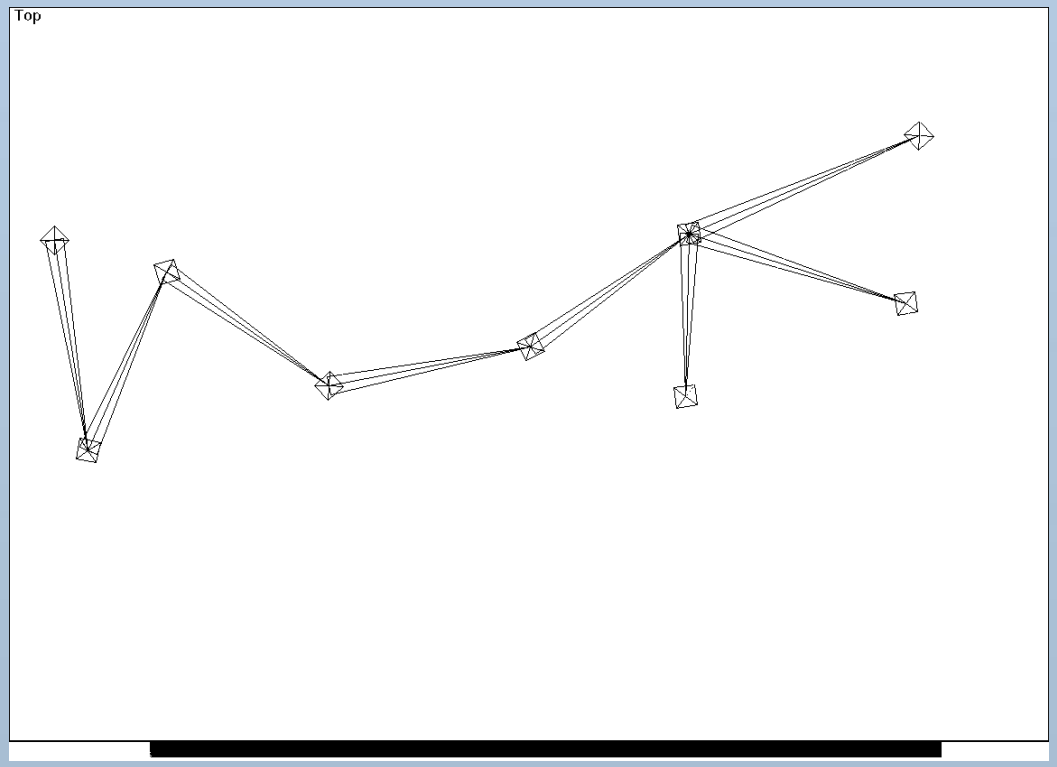
Ejemplo:

```
1: glLoadIdentity();
2: glTranslatef(1,0,0);
3: glRotatef(45,0,0,1);
4: DrawObject();
```



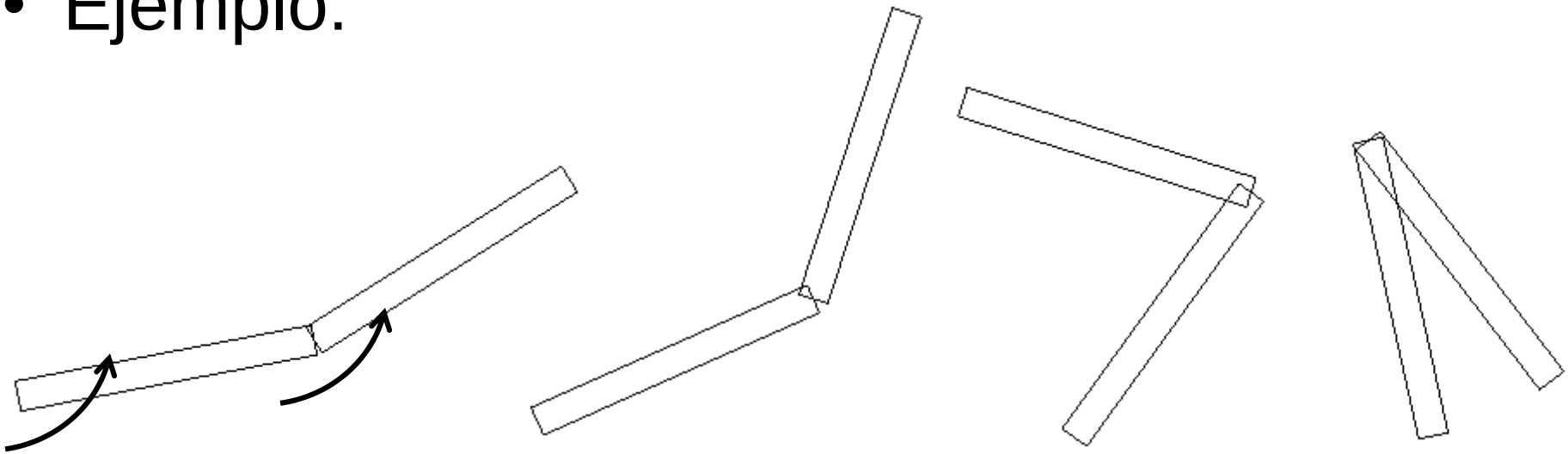
Transformaciones

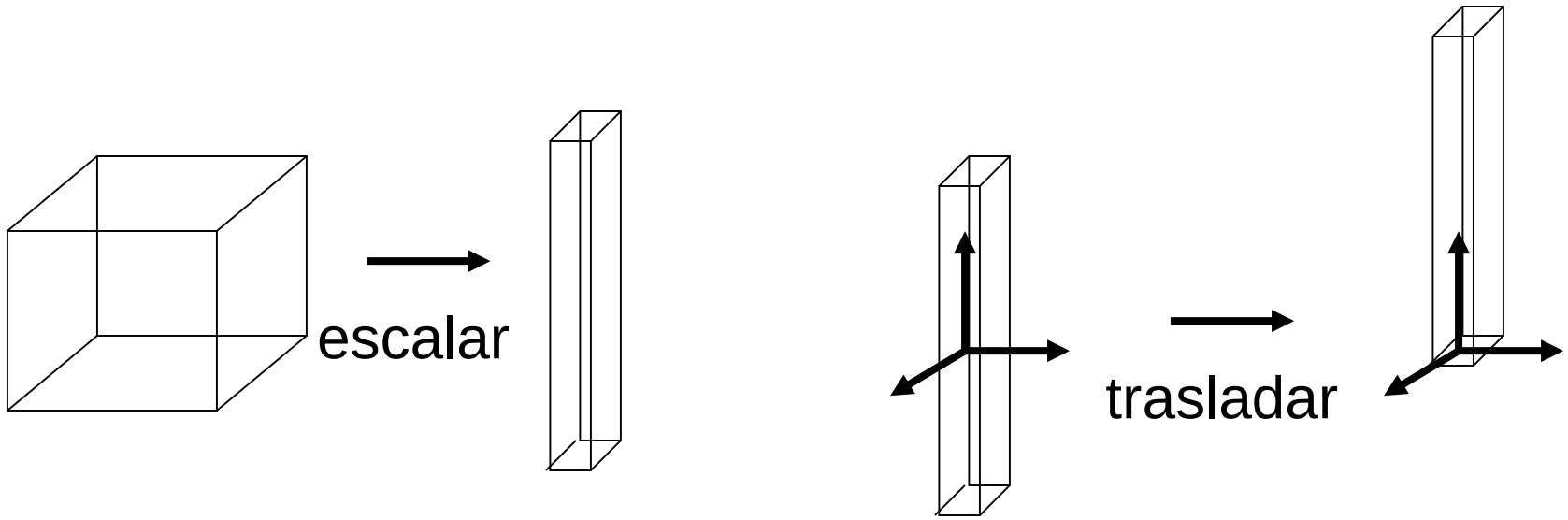
- La segunda alternativa es más útil para modelado jerárquico (articulaciones en general)



- La primera alternativa es problemática en el caso del escalado. El orden incorrecto puede traer resultados no deseados

- Ejemplo:





Transformaciones

TECNOLÓGICO DE MONTERREY.

Ejemplo

```
glLoadIdentity();          //Al Origen
glRotatef(r1,0,0,1);       //El segmento principal
glTranslatef(0.5,0,0);     //Mover el pivote al origen
glScalef(1,0.1,0.1);       //Escalarlo
glutWireCube(1.f);         //Dibujar el cubo
glScalef(1,10,10);         //Escalarlo de regreso
glTranslatef(0.5,0,0);     //Al fin del cubo
glRotatef(r2,0,0,1);       //rotar el segundo segmento
glTranslatef(0.5,0,0);     //Mover el pivote al origen
glScalef(1,0.1,0.1);       //Escalar
glutWireCube(1.f);         //y desplegar
r1+=0.1;
r2+=0.2;
```

Viewing Transform

TECNOLÓGICO DE MONTERREY.

Viewing Transformation

- Es aplicar rotaciones, traslaciones y escalado de la cámara
- Las transformaciones de la cámara se deben establecer antes de desplegar cualquier objeto
- Es mejor establecer primero las transformaciones de la cámara y después las transformaciones de Modelview

Viewing Transform

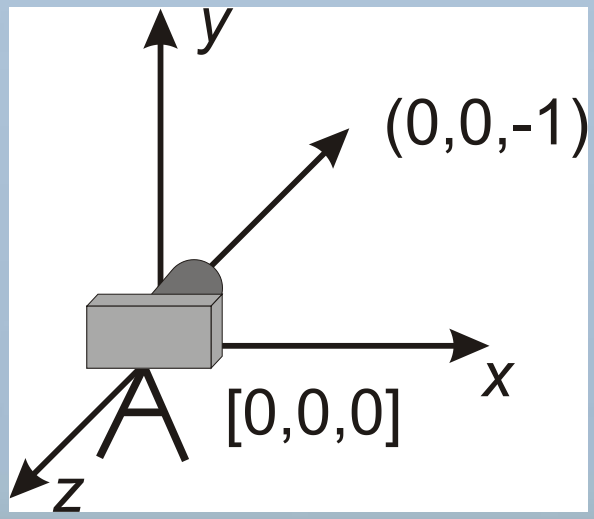
- Hay dos alternativas para crear la noción de movimiento:
  - Mover la camara, Rotar la cámara
  - Mover los objetos en la dirección contrara, Rotar los objetos en el orden contrario
- En una escena lo primero que hay que hacer es posicionar la cámara:

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

posición [0,0,0]

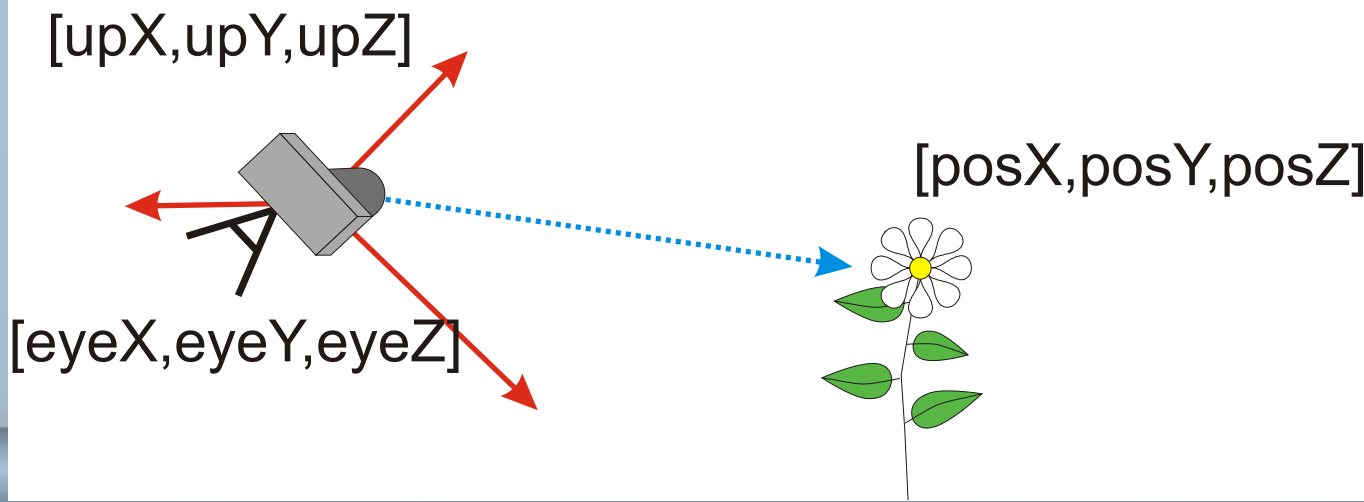
Ver hacia (0,0,-1)



Viewing Transform

- El posicionamiento de la cámara en 3D puede ser complejo
- Este proceso puede ser simplificado usando la función de GLU

```
gluLookAt(
GLdouble eyeX,GLdouble eyeY,GLdouble eyeZ,
GLdouble posX,GLdouble posY,GLdouble posZ,
GLdouble upX,GLdouble upY,GLdouble upZ)
```



Pila de Matrices

- Podemos guardar y recuperar estados de las matrices mediante acciones Push y Pop
- Cuando hacemos un Push, la matriz actual se copia a la pila de matrices
- Cuando hacemos un Pop, la matriz en el tope de la pila se copia a la matriz actual

```
void glPopMatrix()
void glPushMatrix()
```

- La matriz actual es la establecida por glMatrixMode
- La profundidad del stack se puede obtener con:

```
glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH)
glGetIntegerv(GL_MAX_PROJECTION_STACK_DEPTH)
```

Pila de Matrices

TECNOLÓGICO DE MONTERREY.

¿Cuál es el uso de la pila de matrices?

- Hacer las transformaciones inversas es caro en términos de rendimiento
- Guardar la matriz es más barato

Cualquier operación puede ser invertida con:

```
glPushMatrix();
   cualquier_transformacion();
glPopMatrix();
```

Pila de Matrices

TECNOLÓGICO DE MONTERREY.

Ejemplo

```
glPushMatrix;                    //guardar el edo inicial
 glRotatef(r1,0,0,1);            //rotar el seg principal
 glTranslatef(0.5,0,0);                //al origen
 glPushMatrix();
  glScalef(1,0.1,0.1);           //escalarlo
  glutWireCube(1.f);             //desplegar el cubo
 glPopMatrix()                   //escalar de regreso
 glTranslatef(0.5,0,0);          //al final
 glRotatef(r2,0,0,1);            //el 2do segmento
 glTranslatef(0.5,0,0);          //al origen
 glScalef(1,0.1,0.1);            //escalar
 glutWireCube(1.f);              //desplegar
 r1+=0.1;r2+=0.2;
glPopMatrix();                   //regresar al inicial
```

Proyecciones

- El propósito es definir un volúmen de visible (viewing volume)
- Los objetos dentro del volumen son desplegados, los demás son descartados (clipped)

- Las transformaciones de proyección pueden ser
  - Definidas por el usuario
  - Proyección Ortográfica (paralela)
  - Proyección Perspectiva

- La proyección define

  los clipping planes

TECNOLÓGICO DE MONTERREY.

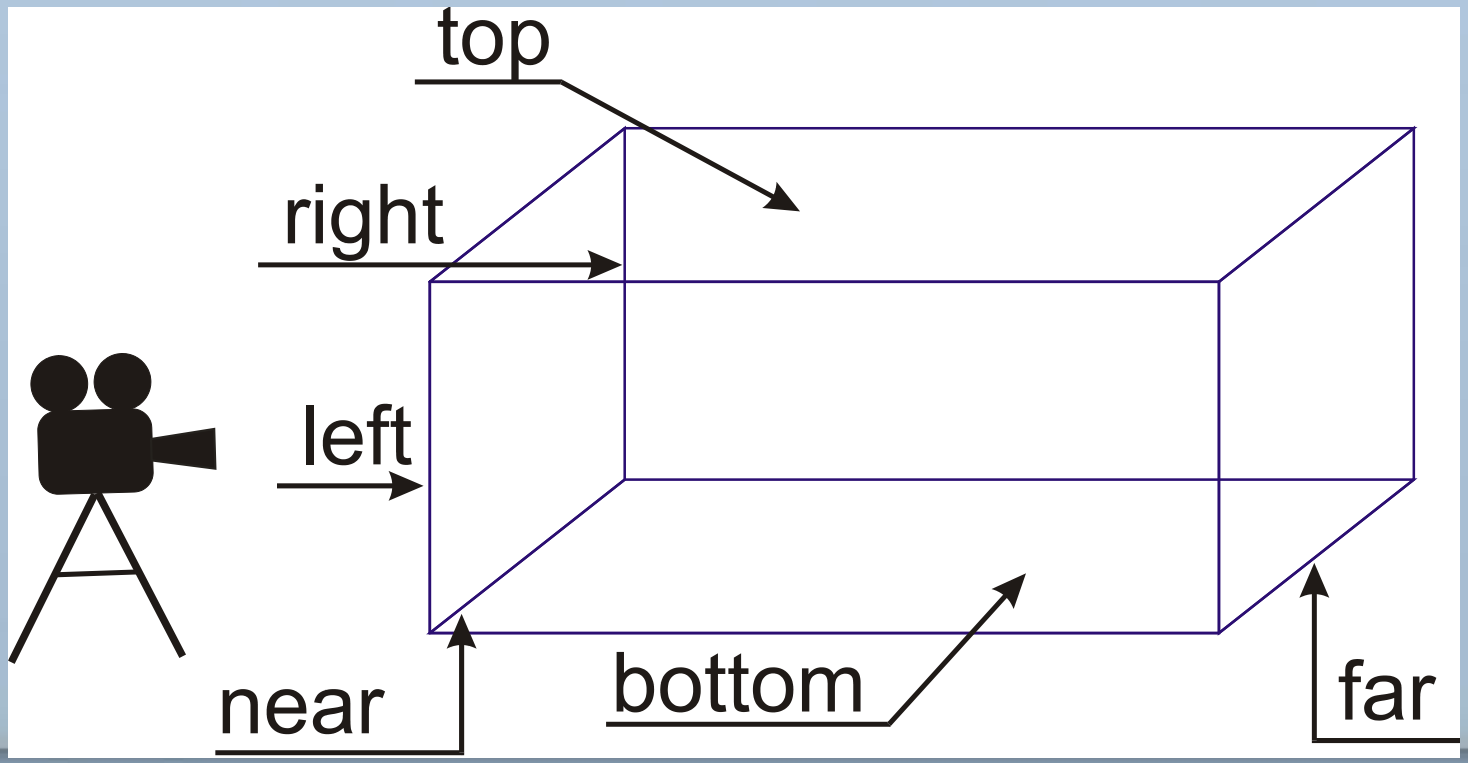
Proyecciones definidas por el usuario

```
glMatrixMode(GL_PROJECTION);
glLoadMatrix(m);
```

- Donde **m** define la matriz de proyección
- Ejemplos: perspectiva de dos puntos, proyecciones oblicuas, isométricas, dimétricas, trimétricas

Projección Ortográfica

```
void glOrtho(GLdouble left,GLdouble right,
             GLdouble bottom,GLdouble top,
             GLdouble near, GLdouble far)
```



Proyecciones

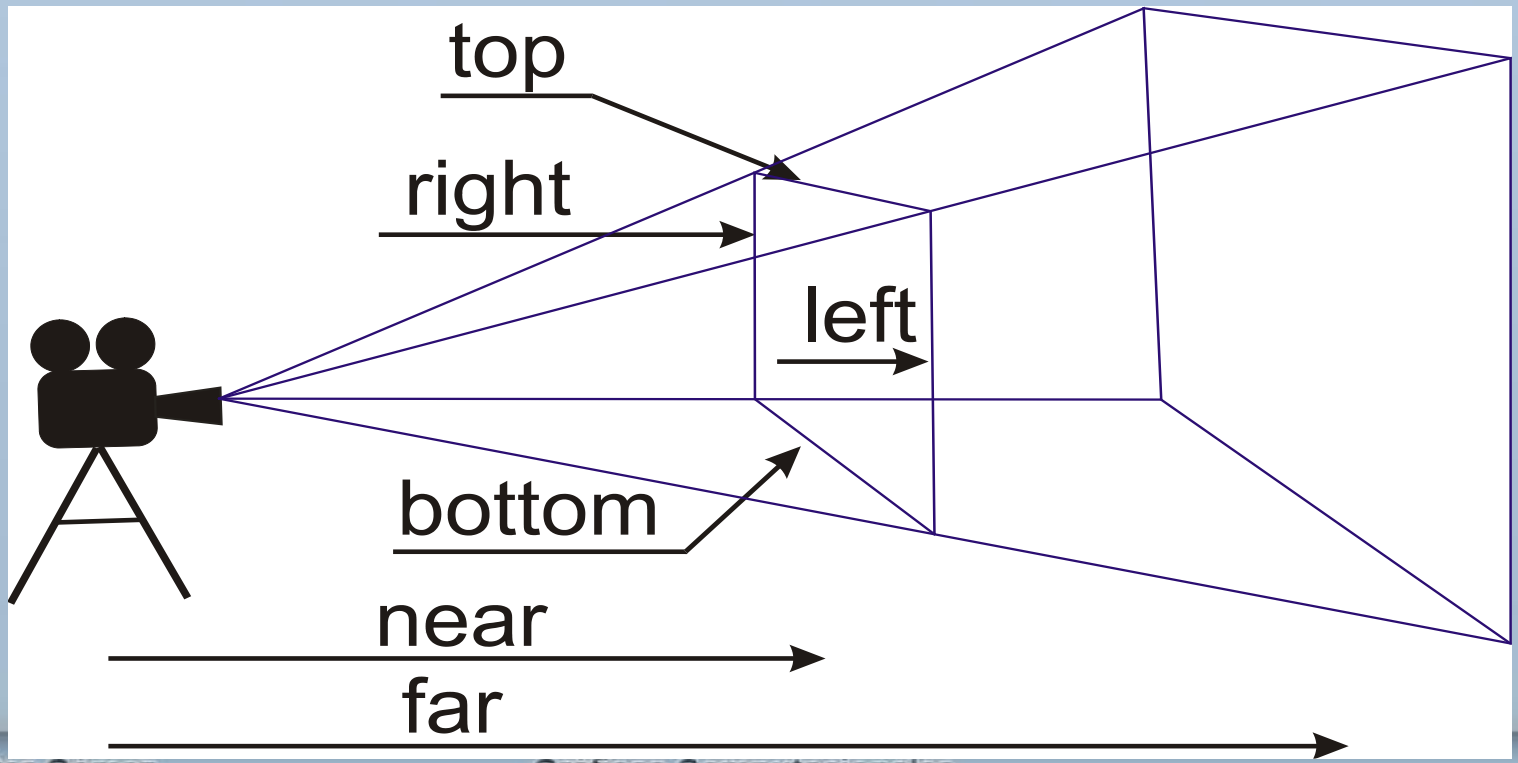
- Las distancias son relativas a la posición de la camara
- La cámara puede estar dentro del volúmen visible (view volume)

  glOrtho(-1,1,-1,1,-1,1)
- La camara puede estar completamente fuera del view volume

  glOrtho(-12,-10,-1,1,-1,1)

Proyección Perspectiva

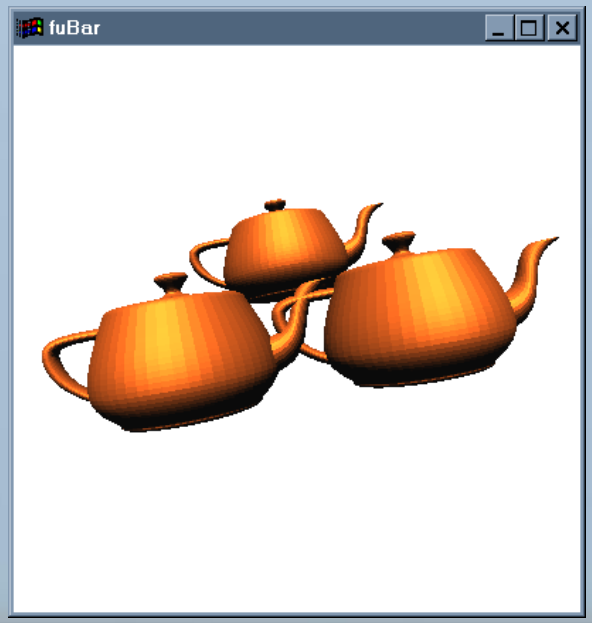
```
void glFrustum(GLdouble left,GLdouble right,
               GLdouble bottom,GLdouble top,
               GLdouble near, GLdouble far)
```

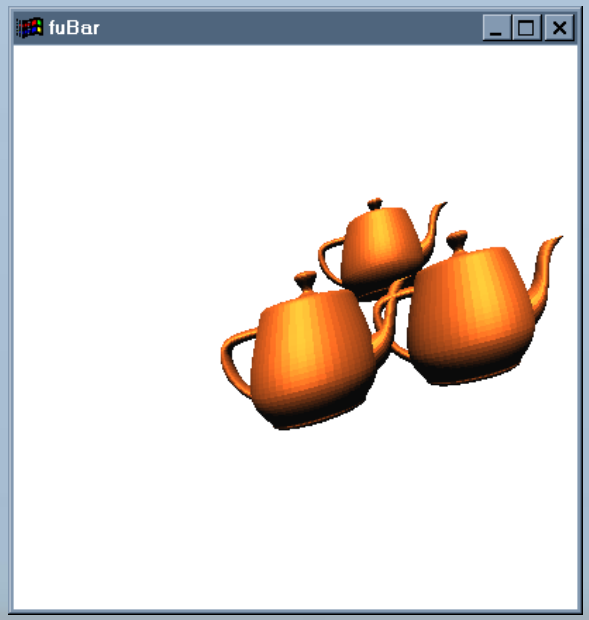


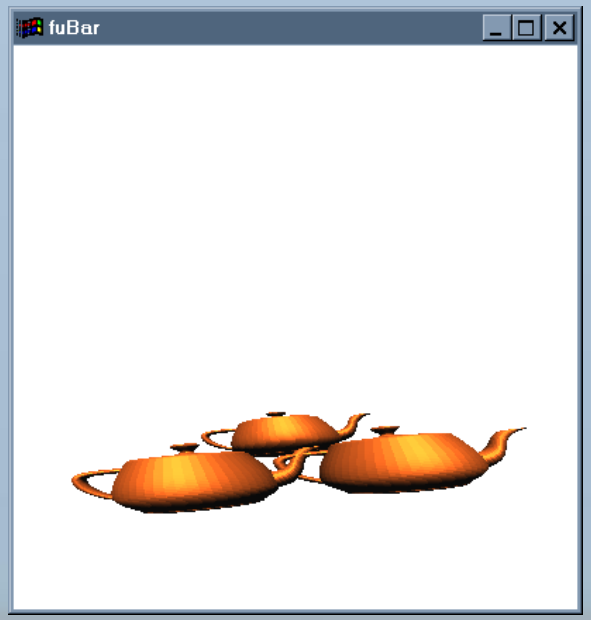
Proyecciones

glFrustum

- No es intuitiva
- Puede ser asimétrica
- Muy confusa



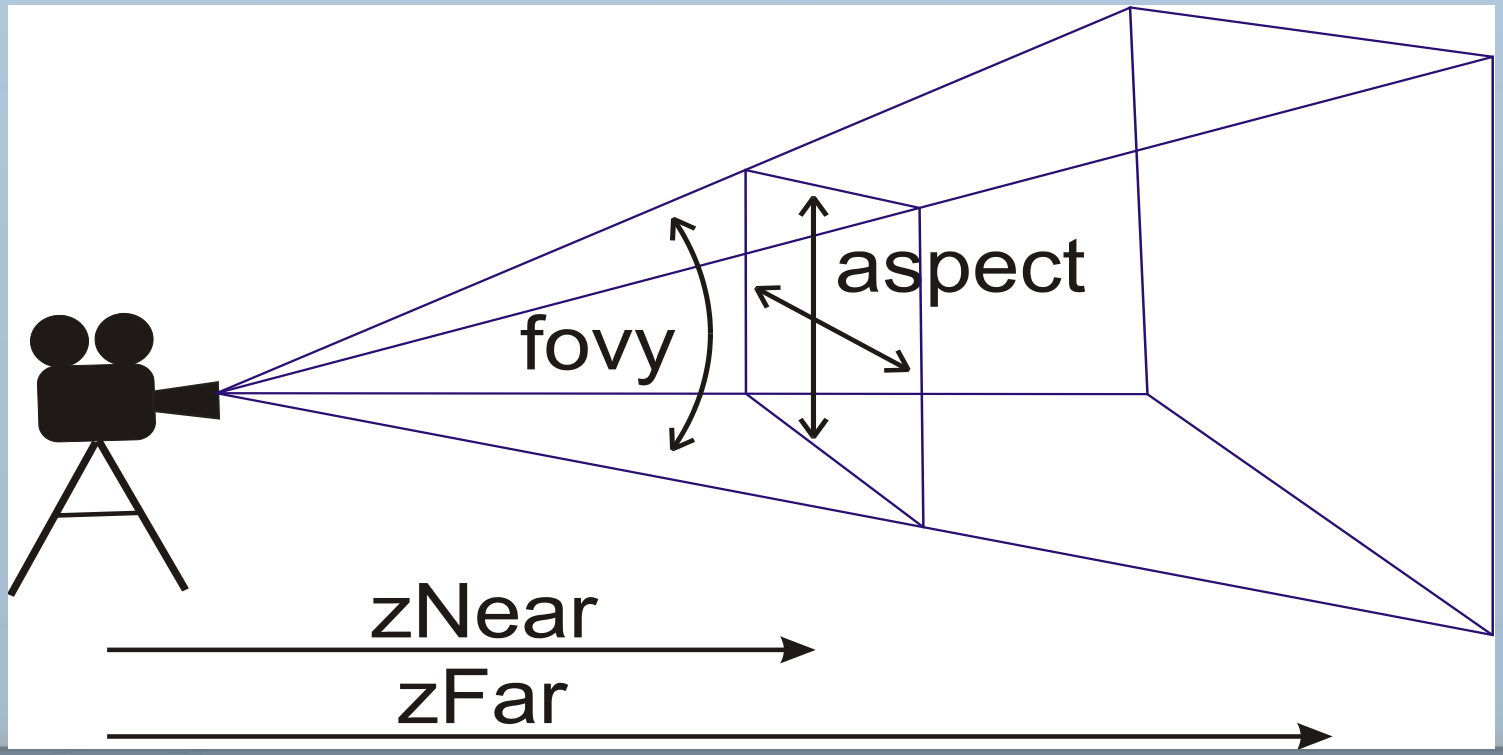




TECNOLÓGICO DE MONTERREY.

Proyección Perspectiva

```
void gluPerspective(GLdouble fovy,
                    GLdouble aspect,
        GLdouble zNear, GLdouble zFar)
```



Proyecciones

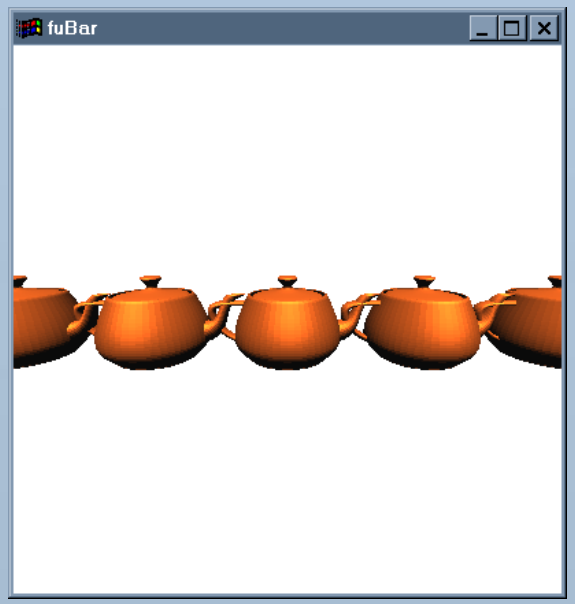
gluPerspective

`fovy` es el campo visual en el eje y (field of view)

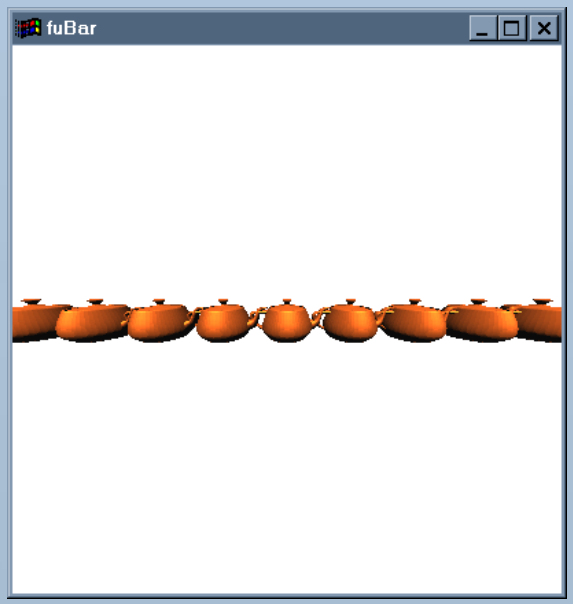
`aspect` es la proporción del viewport actual

No puede ser asimétrica

fovy=60



90



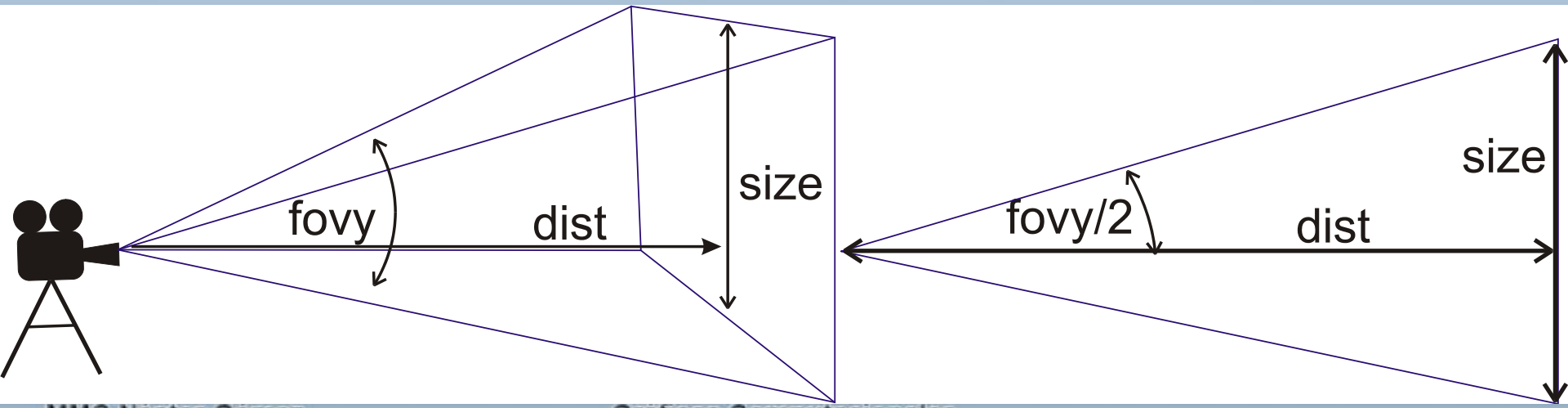
120

Proyecciones

gluPerspective

Podemos definir fovy sabiendo la distancia del objetivo y la altura que deseamos para el encuadre

```
GLdouble FOVY(GLdouble size,GLdouble dist){
 GLdouble theta;
 theta = 2.0*atan(size/2.0, dist);
 return(180.0*theta/ π);
}
```

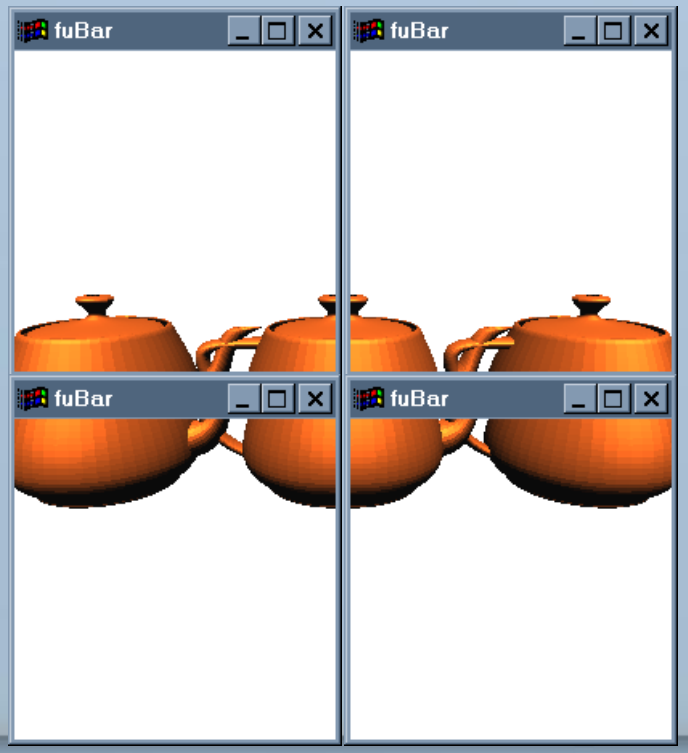


Proyecciones

¿Cómo podemos obtener una imagen del doble del tamaño máximo?

1. Despliega la imagen 4 veces estableciendo las proyecciones como se muestra
2. Salvar las imágenes
3. Juntarlas

x=[-1, 0] y=[0, 1]	x=[0, 1] y=[0, 1]
x=[-1, 0] y=[-1, 0]	x=[0, 1] y=[-1, 0]



Proyecciones

- Código para imagen más grande

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1,0,0,1,    1,10); //upper left
Render();
glLoadIdentity();
glFrustum(0,1,0,1,     1,10);//upper right
Render();
glLoadIdentity();
glFrustum(-1,0,-1,0,   1,10);//lower left
Render();
glLoadIdentity();
glFrustum(0,1,-1,0,    1,10);//lower right
Render();
```

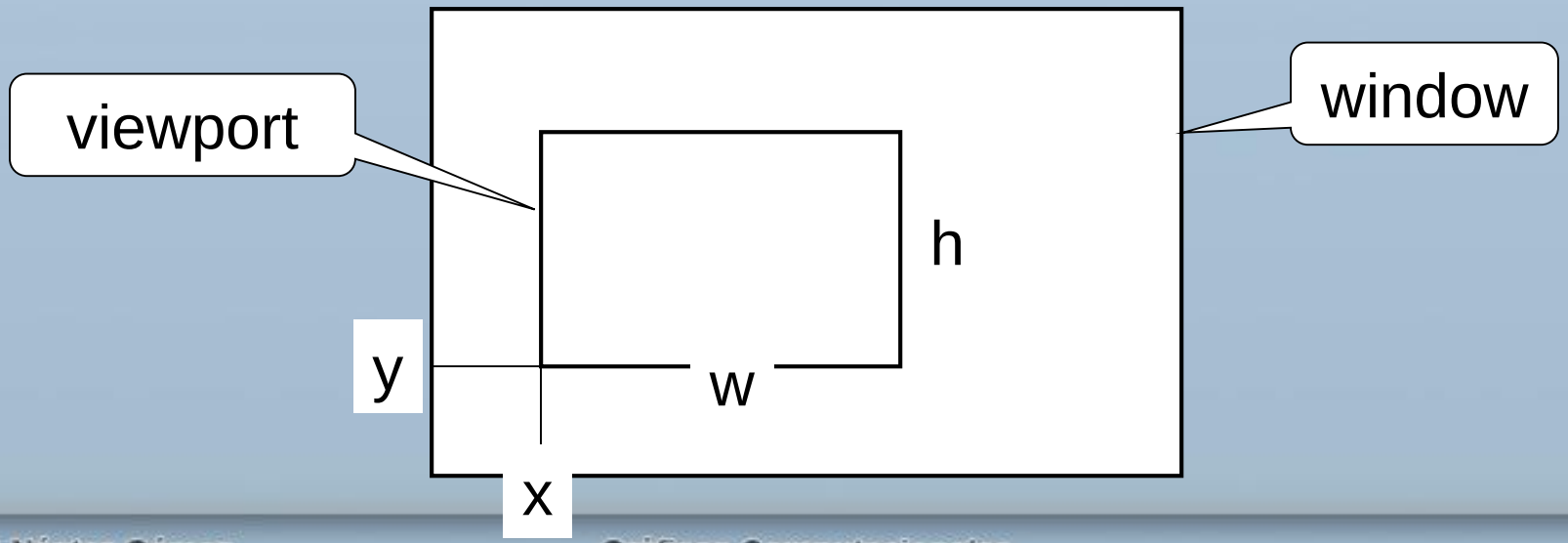
Viewport

Viewport es la porción de la ventana que es usada para el render

```
void glViewport(GLint x,GLint y,
                GLsizei w, GLsizei h)
```

x y **y** especifican la esquina inferior izquierda

w y **h** son las dimensiones del viewport



Viewport

El viewport se establece usualmente en el callback de Reshape

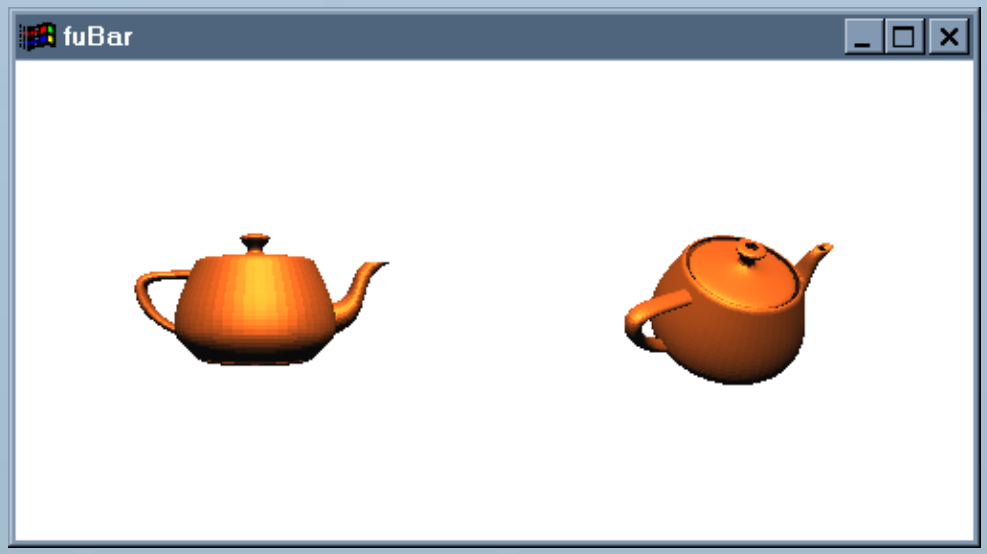
```
void Resize(int w, int h){
  glViewport(0,0,w,h);
…
}

int main(int argc, char **argv){
  glutInit(&argc, argv);
…

  glutDisplayFunc(Display);
  glutReshapeFunc(Resize);
…
}
```

Viewport

Ejemplo

Dos viewports en una ventana

```
int x = 600, y = 300;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0, x/(2.0*y), 1, 10.0);//!!!
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glViewport(0,0,x/2,y);
glutSolidTeapot(0.8);
glViewport(x/2,0,x/2,y);
glRotatef(60, 1, 1, 0);
glutSolidTeapot(0.8);
```



Clipping Planes

El usuario puede definir sus propios clipping planes

```
void glClipPlane(GLenum plane,
                 const GLdouble *eqn)
```

Plane es **GL_CLIP_PLANE0,..., GL_CLIP_PLANE5**

eqn es un apuntador a cuatro doubles A,B,C,D y Ax+By+Cz+D=0 define el plano

Los puntos que satisfagan la condición:

$(A\ B\ C\ D)M^{-1}(x\ y\ z\ w)^T>=0$

no son desplegados (M es la matriz actual de modelview)

Los planos de deben de activar con:

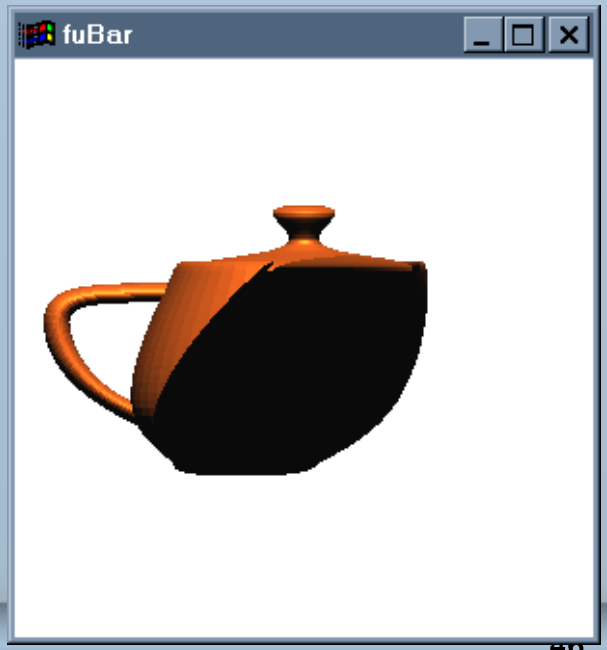
glEnable(GL_CLIP_PLANE1)

Clipping Planes

TECNOLÓGICO DE MONTERREY.

Ejemplo:

```
GLdouble eqn[]={-1.0,1.0,-1.0,0.0};

glLoadIdentity();
glTranslatef(0,0,-5);
glClipPlane(GL_CLIP_PLANE0,eqn);
glEnable(GL_CLIP_PLANE0);
glutSolidTeapot(1.5);
```



Visión Estereo

- GLUT provee herramientas para la visión estereo
- En ocasiones se necesita más control de que provee GLUT
- OpenGL soporta varios buffers para render
- Para renderear un frame en estereo:
  - Un dispositivo que lo soporte
  - El ojo derecho/izquierdo debe renderear en el back buffer derecho/izquierdo
  - Los back buffers se deben desplegar correctamente

Rendering Buffers

OpenGL soporta varios rendering buffers

Para cambiar el rendering buffer activo

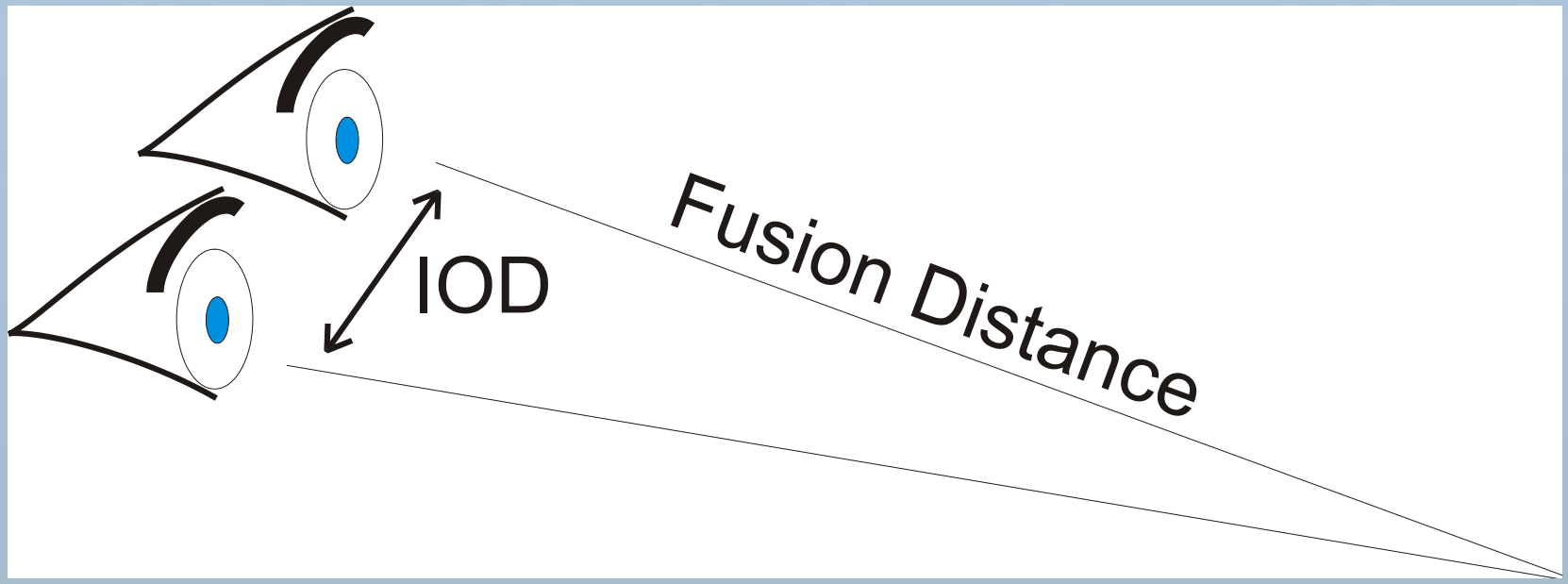
```
void glDrawBuffer(GLenum mode)
void glReadBuffer(GLenum mode)
```

mode es `GL_NONE`, `GL_FRONT_LEFT`, `GL_FRONT_RIGHT`, `GL_BACK_LEFT`, `GL_BACK_RIGHT`, `GL_FRONT`, `GL_BACK`, `GL_LEFT`, `GL_RIGHT`, `GL_FRONT_AND_BACK`, `GL_AUXi`

Visión Estéreo

Se debe establecer dos valores:

- Distancia Interocular, IOD (interocular distance)
- Distancia de Fusión



Visión Estereo

TECNOLÓGICO DE MONTERREY.

```
//left back buffer
glPushMatrix();glDrawBuffer(GL_BACK_LEFT);
gluLookAt(   -IOD/2.0, 0.0, EYE_BACK, //posicion
             0.0, 0.0, 0.0, //foco
             0.0, 1.0, 0.0); //up vector
RenderScene();
//right back buffer
glDrawBuffer(GL_BACK_RIGHT);
gluLookAt(   IOD/2.0, 0.0, EYE_BACK, //posicion
             0.0, 0.0, 0.0, //foco
             0.0, 1.0, 0.0); //up vector
RenderScene();
glutSwapBuffers();
```

Resúmen

- Transformaciones de Coordenadas
- Transformaciones de Modelado
- Transformaciones de la Cámara
- Proyecciones
- Trucos
  - Imágenes grandes
  - Visión Estereo
  - Clipping Planes