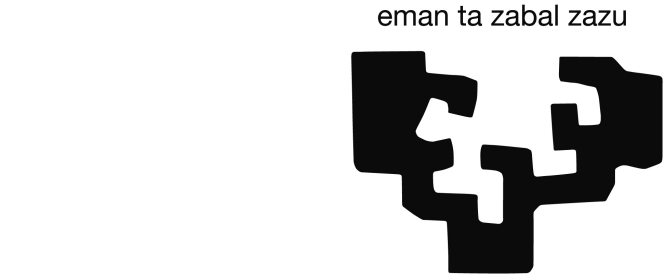
Transformaciones geométricas



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Unai Lizarralde Imaz Noel Arteche Echeverria
Edu Vallejo Arguinzoniz

24 de diciembre de 2018

Resumen

Esta práctica contempla las transformaciones geométricas de objetos tridimensionales y su representación en el espacio

Índice

Índice de figuras

1. Introducción

Esta memoria cubre el planteamiento, solución y documentación de la segunda práctica de la asignatura Gráficos por computador. Se trata de la primera parte de una práctica incremental que, en esta iteración, cubre los conceptos de transformaciones geométricas.

Partimos de una versión inicial del proyecto, que carga objetos tridimensionales en formato .obj, los muestra en pantalla y permite ampliar o reducir la perspectiva. Pueden cargarse varios objetos simultáneamente y cambiar entre el control de cada uno de ellos empleando la tecla TAB.

El trabajo ha mejorado el código existente añadiendo la posibilidad de mover los objetos por el espacio de acuerdo a tres transformaciones geométricas (escalado, traslación y rotación). Estas se llevan a cabo empleando las correspondientes matrices de estas isometrías, que se explican en detalle más adelante. Posteriormente se han añadido reflexiones (respecto a los planos formados por la base canónica) y el efecto *shear mapping*.

2. Transformaciones geométricas

En esta práctica representamos objetos tridimensionales proyectados al plano $\mathbb{R}^2$. Para ello utilizamos objetos representados por puntos y vectores. Estas modificaciones se realizan en el espacio afín y, por tanto, añadiremos una dimension más al objeto tridimensional para proyectar el resultado de las transformaciones y facilitar los cálculos. Las transformaciones geométricas se pueden realizar con cualquier vector del espacio, sin embargo, las transformaciones se han restringido a los vectores canónicos.

Para modificar cada punto de un objeto se le va asignar una matriz de instancia cada vez, es decir, generará la nueva a partir de multiplicar la anterior. La matriz inicial es la matriz identidad para que no realice ninguna transformación de antemano, pero tampoco sea la matriz nula (ya que entonces todos los puntos serían proyectados al origen).

Representación de puntos:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Representación de vectores:

$$\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

Como se puede observar, en el espacio afín asignaremos el valor 1 en la cuarta coordenada de un punto y el valor 0 cuando se trate de un vector.

Otro aspecto de suma importancia radica en la coordenadas locales del objeto y las coordenadas globales del espacio. En el primer caso el objeto siempre estará en el eje de las coordenadas, mientras que en el segundo caso el objeto estará en una posición en concreto dependiendo de las modificaciones que se hayan realizado sobre el mismo. Para modificar el objeto con respecto a las coordenadas globales se multiplica a la matriz de transformación actual por la izquierda, y, por el contrario, si se trata de una modificación con respecto a las variables locales se multiplica por la derecha a la matriz de transformación de ese instante.

2.1. Traslación

La traslación permite el movimiento de un objeto en la dirección del vector y un coeficiente que indique la distancia a recorrer en esa dirección. Los vectores serán los canónicos y el coeficiente estático ($KG\_TRANSLATE\_STEP$). La matriz de traslación se define de la siguiente manera:

$$M_{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde a modifica el punto en el eje x, b modifica el punto en el eje y y c modifica el punto en el eje z. En el caso particular de la práctica, solamente habra que asignar a uno de ellos la distancia a modificar, pues se modifica en el eje de las coordenadas.

2.2. Escalado

El escalado aumenta el tamaño del objeto en la dirección y cantidad indicados. Para cada eje se debe establecer un parámetro que multiplique a cada eje, para aumentarlo según lo deseado, especificado en la variable

global (KG_SCALE_STEP). La matriz de escalado se define de la siguiente manera:

$$M_{\mathrm{E}} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde a multiplica al eje x, b multiplica al eje y y c multiplica al eje z . En este caso, el escalado se realiza sobre un solo eje a la vez y, por tanto, el parámetro del eje a modificar será modificado y a los dos restante se les asigna la unidad para no causar ninguna modificación por ser el elemento neutro de la multiplicación.

2.3. Rotación

La rotación de un objeto consiste en girarlo con respecto a un eje definido mediante un vector en una cantidad de radianes α. Los vectores sobre los que se va a rotar son los vectores canónicos, por lo que para cada eje hay una matriz de rotación en específico. El ángulo de rotación, α, se ha especificado de forma constante mediante la variable global (KG_ROTATE_STEP).

Para rotar sobre cualquier eje w:

$$M_{\mathrm{RT}} = \begin{pmatrix} (\cos\alpha) + (1-\cos\alpha)x^2 & (1-\cos\alpha)xy - z\sin\alpha & (1-\cos\alpha)xz + y\sin\alpha & 0 \\ (1-\cos\alpha)xy + z\sin\alpha & (\cos\alpha) + (1-\cos\alpha)y^2 & (1-\cos\alpha)yz - x\sin\alpha & 0 \\ (1-\cos\alpha)xz - y\sin\alpha & (1-\cos\alpha)yz + x\sin\alpha & (\cos\alpha) + (1-\cos\alpha)z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para rotar sobre el eje x:

$$w = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RT}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para rotar sobre el eje y:

$$w = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RT}} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para rotar sobre el eje z:

$$w = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RT}} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A partir de la matriz de rotación general se puede deducir una matriz de rotación para cada eje en particular.

2.4. Reflexión

La reflexión es un reposición de los puntos de un objeto contenidos en una partición del espacio definido por un hiperplano a la partición opuesta de forma que los puntos están a la misma distancia del plano. Es decir, es una isometría vectorial en el que los puntos quedan trasladados de forma ortogonal respecto a un hiperplano a la misma distancia que al comienzo.

Los planos para realizar la reflexión del objeto son $x = 0$, $y = 0$ y $z = 0$. Para la reflexión hemos utilizado las matrices reflectoras de Housenholder que nos permiten trasladar cada punto en la dirección del vector normal del plano.

Sea un plano cualquiera con el vector perpendicular siguiente en el espacio afín:

$$u = \begin{pmatrix} u_{\mathrm{x}} \\ u_{\mathrm{y}} \\ u_{\mathrm{z}} \\ 0 \end{pmatrix}$$

La reflexión del objeto con respecto a dicho plano se obtiene mediante la matriz siguiente:

$$M_{\mathrm{RF}} = \begin{pmatrix} 1-2u_x^2 & -2u_xu_y & -2u_xu_z & 0 \\ -2u_xu_y & 1-2u_y^2 & -2u_yu_z & 0 \\ -2u_xu_z & -2u_yu_z & 1-2u_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sin embargo ya conocemos los planos a los cuales se va a efectuar la reflexión, por ello, se van a especificar matrices para cada vector perpendicular concreto con respecto a cada plano.

Para el plano $x = 0$:

$$u = \begin{pmatrix} u_{\mathrm{x}} \\ u_{\mathrm{y}} \\ u_{\mathrm{z}} \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RF}} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para el plano $y = 0$:

$$u = \begin{pmatrix} u_{\mathrm{x}} \\ u_{\mathrm{y}} \\ u_{\mathrm{z}} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RF}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para el plano $z = 0$:

$$u = \begin{pmatrix} u_{\mathrm{x}} \\ u_{\mathrm{y}} \\ u_{\mathrm{z}} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$M_{\mathrm{RF}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

De esta manera, se logra la reflexión conociendo un vector perpendicular del plano.

2.5. Shearing

El *shear mapping* consiste en deformar el objeto manteniendo una de las coordenadas de cada punto del objeto sin modificar y modificando las coordenadas restantes por un coeficiente por la coordenada estable. El *shear mapping* se realiza entonces modificando en base a tres distintos casos que repercutirán en la forma del objeto, similar a torcerlo o estirarlo por dos de sus extremidades.

Se necesitan matrices que establezcan las nuevas coordenadas para cada punto para cada caso y dos coeficientes que vayan a indicar el grado de modificación a y b.

- Se mantiene la coordenada x del objeto:

$$M_{\mathrm{S}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Se mantiene la coordenada y del objeto:

$$M_{\mathrm{S}} = \begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Se mantiene la coordenada z del objeto:

$$M_{\mathrm{S}} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Como se puede observar finalmente, una de las coordenadas quedará sin ninguna perturbación, mientras que las otras dos lo harán en función de los parámetros a y b y el propio valor de la coordenada persistente.

3. Código-C

El código en C se ha desarrollado a partir de la plantilla inicial de trabajo y se ha completado para actuar de acuerdo a las necesidades que las transformaciones exigen en el espacio afín.

3.1. Organización

El código fuente se puede encontrar en el subdirectorio "src", existen varios ficheros, cada uno de los cuales tiene un único fin bien definido.

3.1.1. main.c

Contiene la función de entrada a la aplicación **int main()** y algunas definiciones. La función de entrada se encarga de inicializar la maquina de estados de OpenGL y algunas variables adicionales para que cumplan las invariantes deseadas.

3.1.2. io.c

Contiene las subrutinas de entrada y salida que tratan las interacciones con el usuario. Estas son: la función que decide que hacer cuando se presiona una tecla en la ventana **void keyboard(...)** y el código que permite usar la terminal para indicar un fichero a la hora de cargar modelos.

3.1.3. load_obj.c

Contiene el código necesario para cargar un modelo desde un fichero dado. La función que implementa esta operación se asegura de que el objeto no queda en un estado invalido y actualiza la escena.

3.1.4. display.c

Contiene las funciones que permiten dibujar la escena cuando esta se actualiza.

3.1.5. transform.c

Contiene las funciones que permiten transformar los objetos en escena. La función principal es **void transform(AXIS direction)** que dado un eje transforma el objeto seleccionado de acuerdo con el modo de transformación en el que se esta ahora (traslación, rotación, escalado, reflexión, *shear*). También contiene funciones para hacer zoom y para hacer y deshacer transformaciones.

3.1.6. headers

Por cada fichero '.c' (excepto main.c) hay un fichero '.h' en la carpeta 'include' que declara las funciones que se exportan de cada fichero de código

fuente. Ademas de estos ficheros, existe un header adicional, 'definiciones.h', que como su nombre indica se encarga de definir las diferentes estructuras de datos y constantes que se utilizan a lo largo del resto de ficheros.

3.2. Implementación de las transformaciones geométricas

El código en C refleja las transformaciones y el proceso de dibujar los objetos, así como los inputs que funcionan como interfaz de usuario para realizar las mismas. Se van a incluir partes del código esenciales para su ejecución.

Una de las estrategias principales para resolver los problemas de isometrías, ha sido utilizar una función que genera una matriz de identidad y después modificar la misma para cada transformación concreta.

```
void matIden(GLfloat* mat){
  for (int i = 0 ; i < 16 ; i++) {
    mat[i] = ((i % 4) == (i / 4)) ? 1 : 0;
  }
}
```

También se ha decidido implementar una función para multiplicar matrices, prescindiendo de la función ya ofrecida por OpenGL.

```
void MatMul(GLfloat* mat1, GLfloat* mat2 , GLfloat* mat3){

  int i;
  int j;
  int k;

  nulMat(mat3);
  for (i = 0 ; i < 4 ; i++){
    for (j = 0 ; j < 4 ; j++){
      for (k = 0 ; k < 4 ; k++){
        mat3[4*i + j] += mat1[4*k + j]*mat2[4*i + k];
      }
    }
  }
}
```

Al hacerlo de esta manera, la matriz que multiplica a cada punto del objeto solo hace falta cargarla a la hora de dibujar.

```
glLoadMatrixf(aux_obj->undoStack->mat);
```

Otra decisión que se percibe en la implementación sería la decisión de ofrecer la opción de deshacer una acción y poder rehacerla. Si tras deshacer algún cambio se realiza alguna transformación, no se podrán rehacer los cambios deshechos.

```
void undo(){
  if (_selected_object->undoStack->next != 0){
    struct matStack * ms = _selected_object->undoStack;
```

```
    _selected_object->undoStack = ms->next;
    ms->next = _selected_object->redoStack;
    _selected_object->redoStack = ms;
  }
}

void redo(){
  if (_selected_object->redoStack != 0){
    struct matStack * ms = _selected_object->redoStack;
    _selected_object->redoStack = _selected_object->redoStack->next;
    ms->next = _selected_object->undoStack;
    _selected_object->undoStack = ms;
  }
}
```

Para indicar si se desea hacer una modificación en términos positivos (aumentar el tamaño en el escalado, desplazar el objeto en aumento de coordenada en la traslación, α positivo en la rotación) o no se utiliza el parámetro de entrada *direction* determinado por el input del teclado para conocer cual es la opción que se ha tomado para las transformaciones geométricas.

Así mismo, el teclado tiene comandos para brindar la oportunidad de hacer las transformaciones deseadas. La interfaz funciona mediante comandos definidos en el enunciado, no obstante, la reflexión y *shear mapping* no están contemplados, pues no se pedía implementar ninguno de ellos. Los comandos quedan especificados de la siguiente manera:

- Reflexión (Tecla 'R' o 'r')
  Eje x: Flechas izquierda y derecha
  Eje y: Flechas arriba y abajo
  Eje z: AVPAG y REPAG

- *shear mapping* (Tecla 'S' o 's')
  Eje x: Flechas izquierda y derecha
  Eje y: Flechas arriba y abajo
  Eje z: AVPAG y REPAG

3.2.1. Traslación

```
void translate(int direction, GLfloat * mat){
  matIden(mat);
  if (direction > 0)
    mat[11 + direction] = KG_TRANSLATE_STEP;
  else
    mat[11 - direction] = -KG_TRANSLATE_STEP;

}
```

3.2.2. Escalado

```
void scale(int direction, GLfloat * matrix) {

  matIden(matrix);

  if (direction > 0)
    matrix[5 * (direction - 1)] = KG_SCALE_STEP;
  else
    matrix[5 * ((-direction) - 1)] = 1/KG_SCALE_STEP;
}
```

3.2.3. Rotación

```
 void rotate(int direction, GLfloat* matrix)
{
    double sense;
    sense = direction > 0 ? 1 : -1;
    direction = direction > 0 ? direction : -direction;
    static double seno = sin(KG_ROTATE_STEP);
    static double coseno = cos(KG_ROTATE_STEP);
    matIden(matrix);

    switch(direction) {

            case (1):
                    matrix[5]=coseno;
                    matrix[6]=sense*seno;
                    matrix[9]=-(sense)*seno;
                    matrix[10]=coseno;
          break;

            case(2):
                    matrix[0]=coseno;
                    matrix[2]=-(sense)*seno;
                    matrix[8]=sense*seno;
                    matrix[10]=coseno;
          break;

            case(3):
                    matrix[0]=coseno;
                    matrix[1]=sense*seno;
                    matrix[4]=-(sense)*seno;
                    matrix[5]=coseno;
          break;
    }
}
```

3.2.4. Reflexión

```
void reflect(int direction, GLfloat * matrix)
{
```

```
  matIden(matrix);
    direction = (direction > 0) ? direction : -direction;
    matrix[5*(direction - 1)] *= -1;
}
```

3.2.5. *Shear mapping*

```
void shear(int direction, GLfloat * matrix) {

  matIden(matrix);

  switch(direction) {

    /* positive X */
    case(2):
      matrix[1] = KG_SHEAR_STEP;
      matrix[2] = KG_SHEAR_STEP;
      break;
    /* negative x */
    case(-2):
      matrix[1] = -KG_SHEAR_STEP;
      matrix[2] = -KG_SHEAR_STEP;
      break;
    /* positive y */
    case(1):
      matrix[4] = KG_SHEAR_STEP;
      matrix[6] = KG_SHEAR_STEP;
      break;
    /* negative y */
    case(-1):
      matrix[4] = -KG_SHEAR_STEP;
      matrix[6] = -KG_SHEAR_STEP;
      break;
    /* positive z */
    case(3):
      matrix[8] = KG_SHEAR_STEP;
      matrix[9] = KG_SHEAR_STEP;
      break;
    /* negative z */
    case(-3):
      matrix[8] = -KG_SHEAR_STEP;
      matrix[9] = -KG_SHEAR_STEP;
      break;
    default:
      break;
  }

}
```

4. Muestras en ejecución

Para demostrar el funcionamiento satisfactorio de nuestro programa se va a proceder a mostrar casos representativos de transformaciones geométricas

que hemos implementado. Se van a mostrar tres ejecuciones que van a mostrar distintos casos y como se adapta nuestro programa de forma consecuente.

4.1. Caso primero

La primera prueba se va a realizar mediante el objeto «abioia.obj», y se va a entremezclar para poner a prueba la consistencia de las transformaciones en nuestro programa.

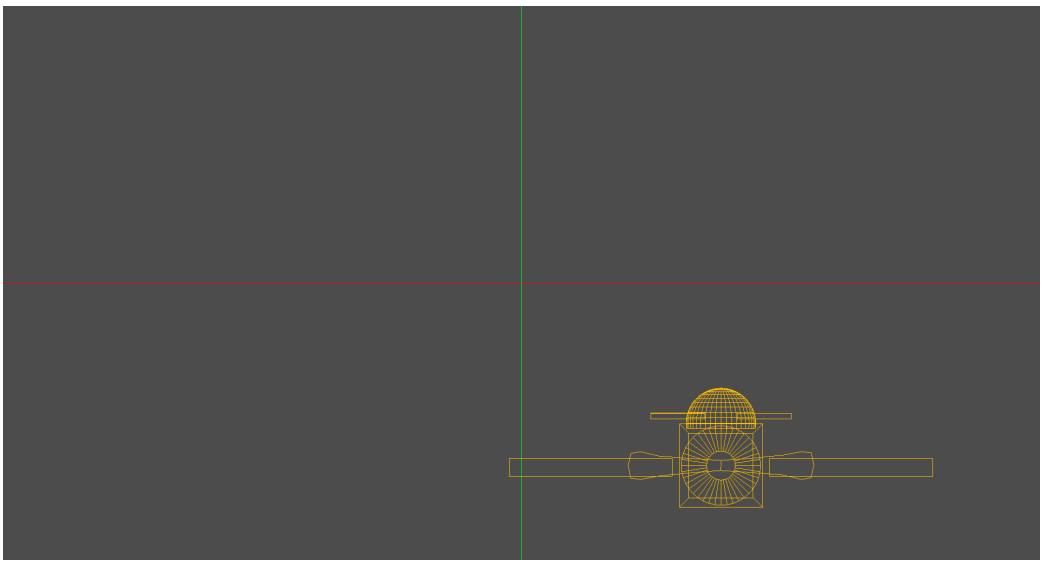


Figura 1: Traslación a una zona fuera del origen de las coordenadas globales

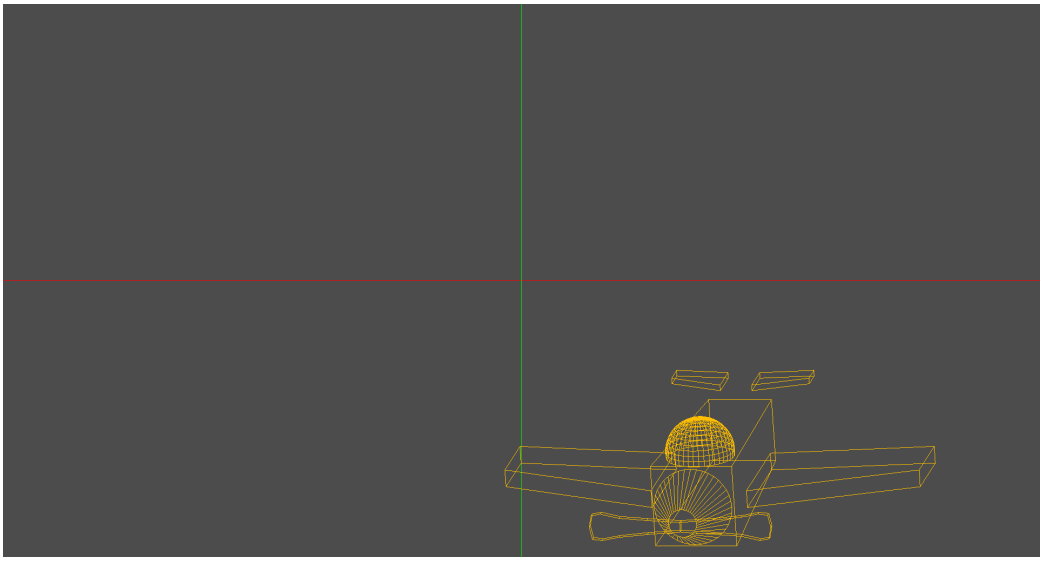


Figura 2: Rotación del objeto en coordenadas locales

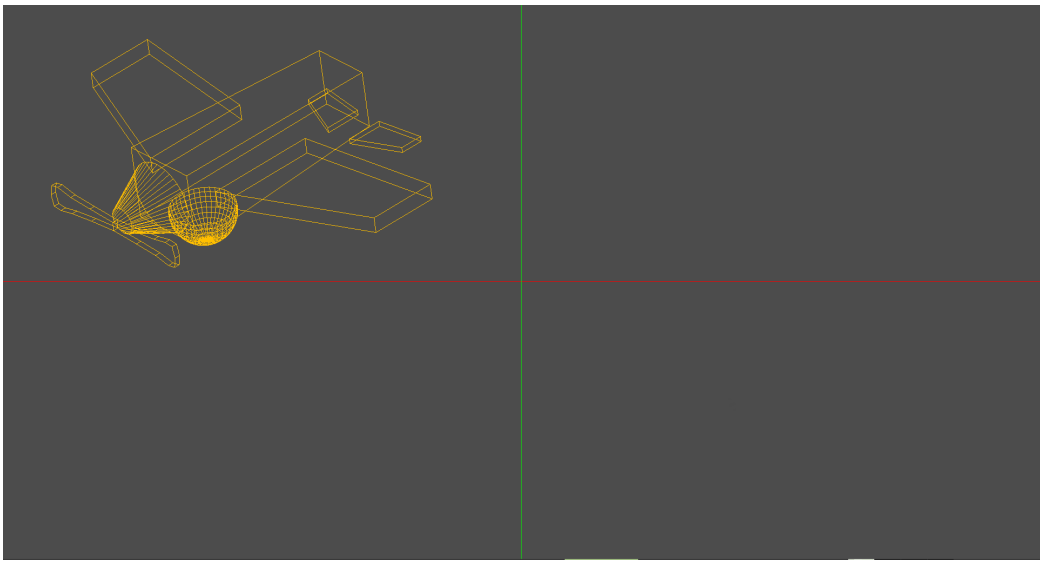


Figura 3: Rotación del objeto en coordenadas globales

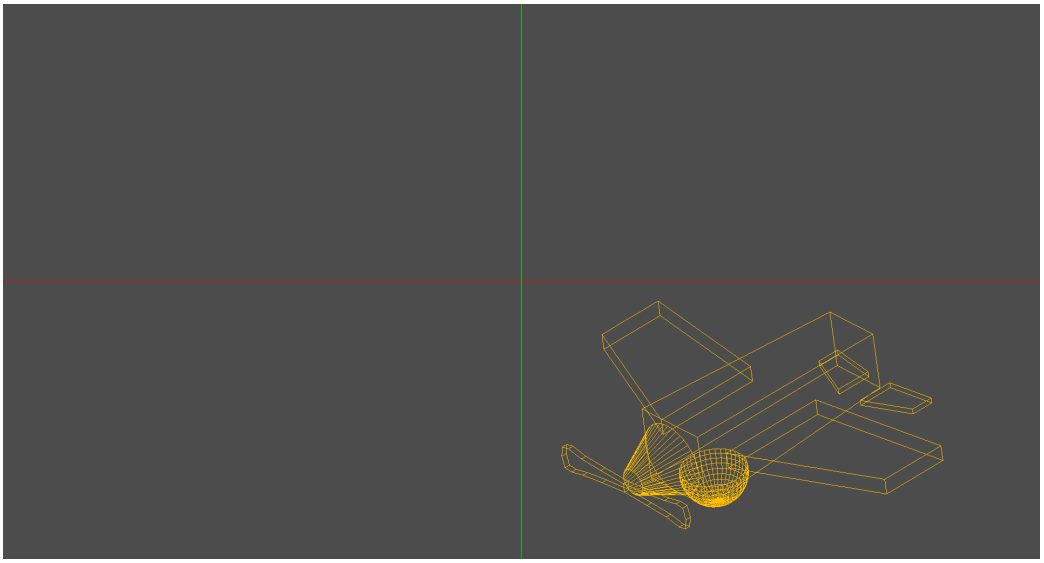


Figura 4: Traslación del objeto en coordenadas locales

4.2. Caso segundo

La segunda prueba consiste en evaluar el escalado fuera del origen de las coordenadas para comprobar si su comportamiento corresponde con la especificada. Comenzaremos tras haberlo colocado en una zona fuera del origen de coordenadas globales.

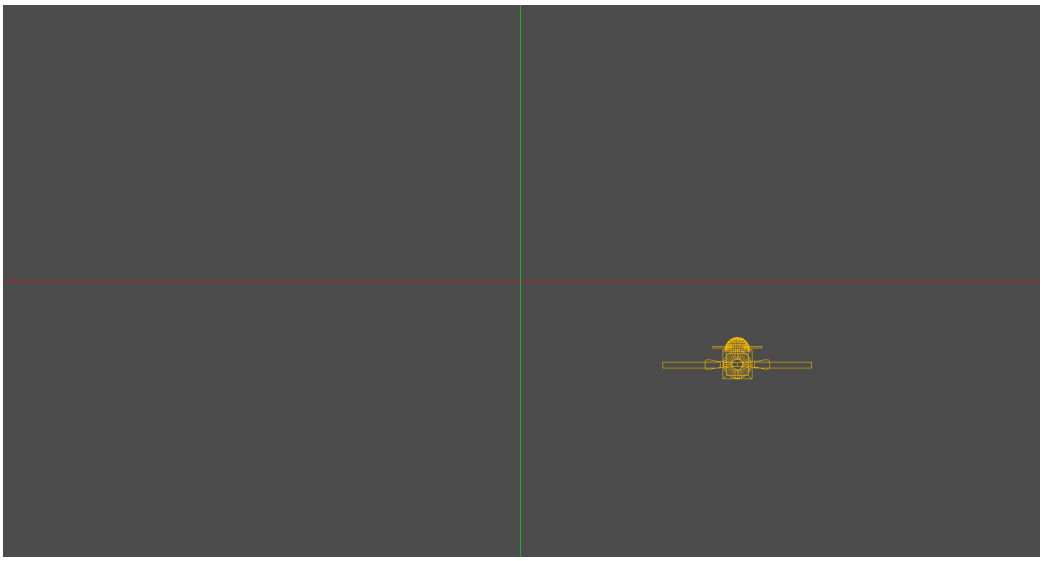


Figura 5: Escalar a un tamaño inferior en coordenadas locales

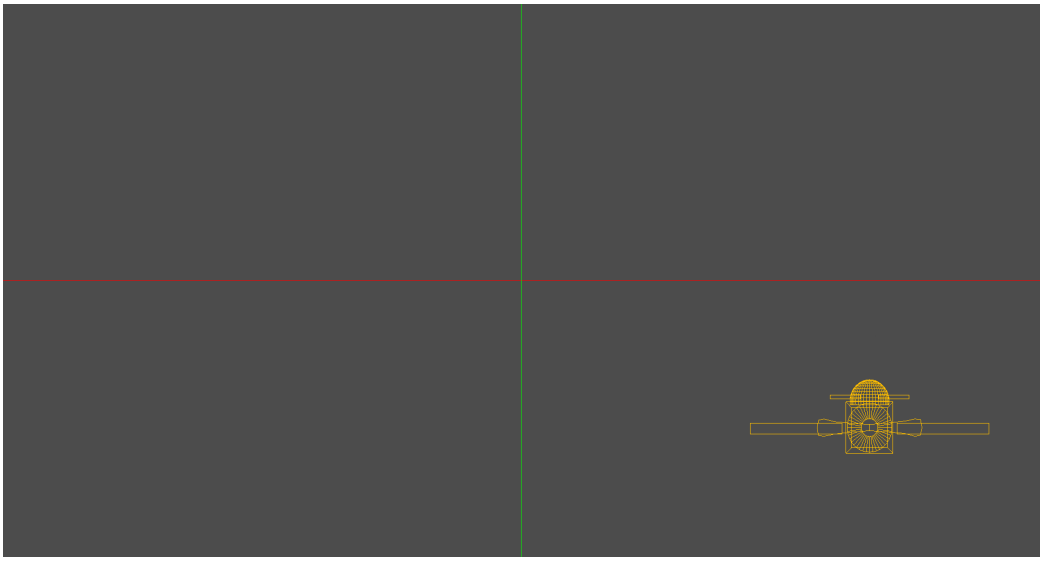


Figura 6: Escalar para aumentar su tamaño en coordenadas globales

4.3. Caso tercero

Para la prueba final se propone hacer una reflexión de un objeto previamente alterado mediante *shear mapping*. He aquí los resultados.

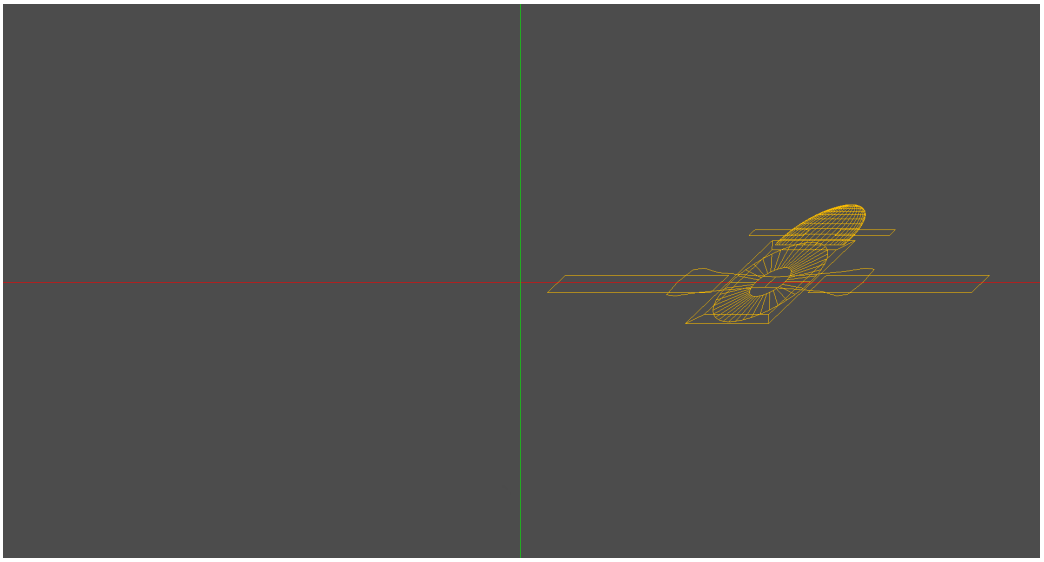


Figura 7: *Shearing* en el eje x de las coordenadas globales

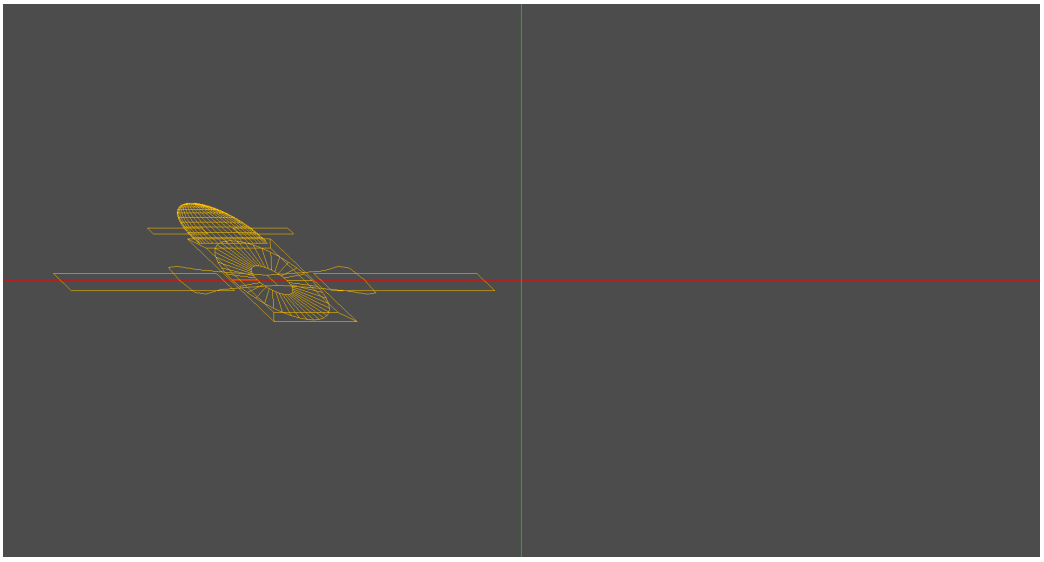


Figura 8: Reflexión del objeto respecto al plano $x = 0$ de las coordenadas globales

5. Conclusiones

Las conclusiones son, en general, positivas. Hemos implementado las transformaciones geométricas básicas (escalado, traslación y rotación) y nos hemos tomado la libertad de añadir dos transformaciones adicionales para ampliar el abanico de opciones: la reflexión y el *shear mapping*.

El programa está correctamente estructurado y permitirá en futuras iteraciones añadir el código correspondiente al control de cámaras y sistemas de iluminación.

Por el momento, no hay problemas de ralentización en la ejecución y hemos intentado evitar cálculos repetidos en la medida de lo posible, aunque, naturalmente, intentaremos optimizar aún más el código en futuras versiones de la práctica.

Por otro lado, queda como propuesta la implementación de unos controles más intuitivos, empleando el ratón en lugar de los atajos del teclado, que harán de la experiencia de usuario algo más amigable.

En general, la práctica ha servido para ver cómo los conceptos teóricos del álgebra lineal y la geometría vectorial (en este caso la utilización de matrices para representar endomorfismos e isometrías en un espacio vectorial) son de gran utilidad en el área de los gráficos por computador.