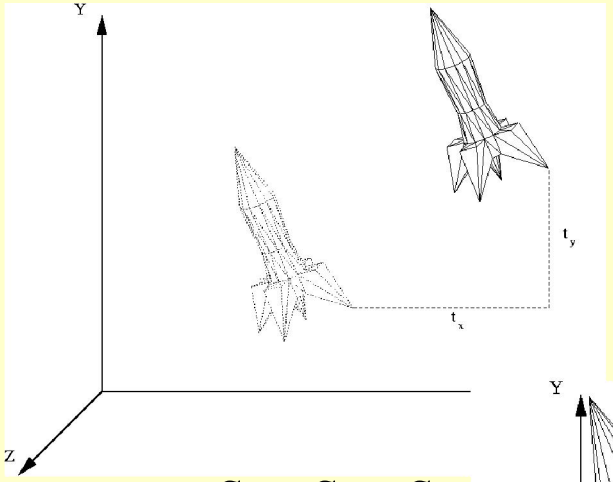
GL transformations-models

- $x' = x+t_x$
- $y' = y+t_y$
- $z' = z+t_z$

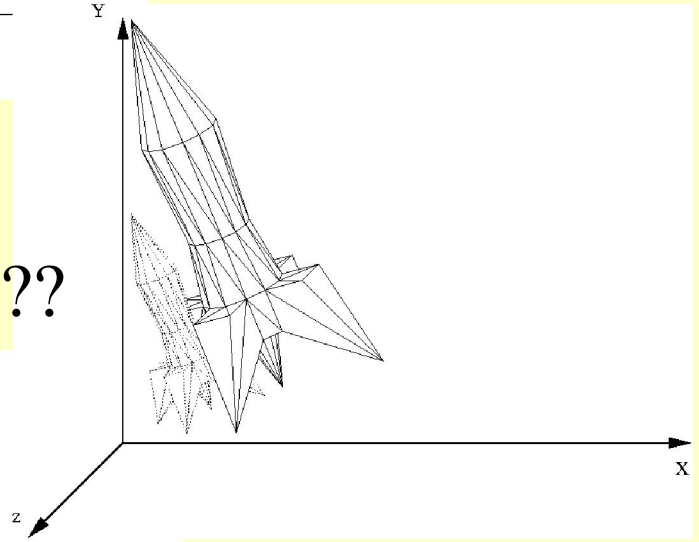
- $1 \quad 0 \quad 0 \quad t_x$
- $0 \quad 1 \quad 0 \quad t_y$
- $0 \quad 0 \quad 1 \quad t_z$
- $0 \quad 0 \quad 0 \quad 1$



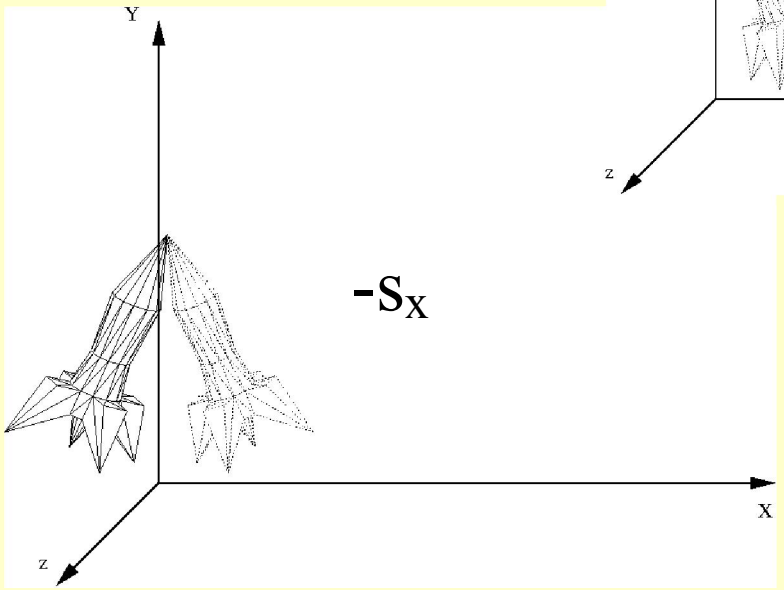
$s_x=s_y=s_z$

uniform

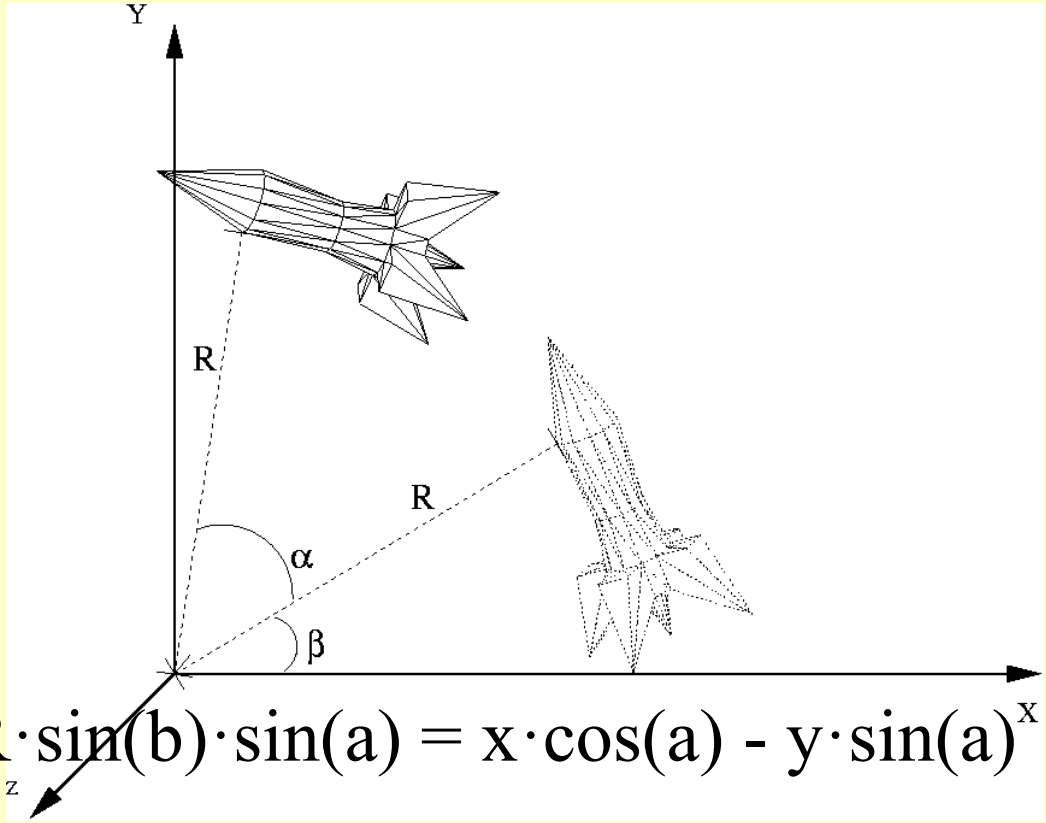
nonuniform?!!??



- $x' = x \cdot s_x$
- $y' = y \cdot s_y$
- $z' = z \cdot s_z$



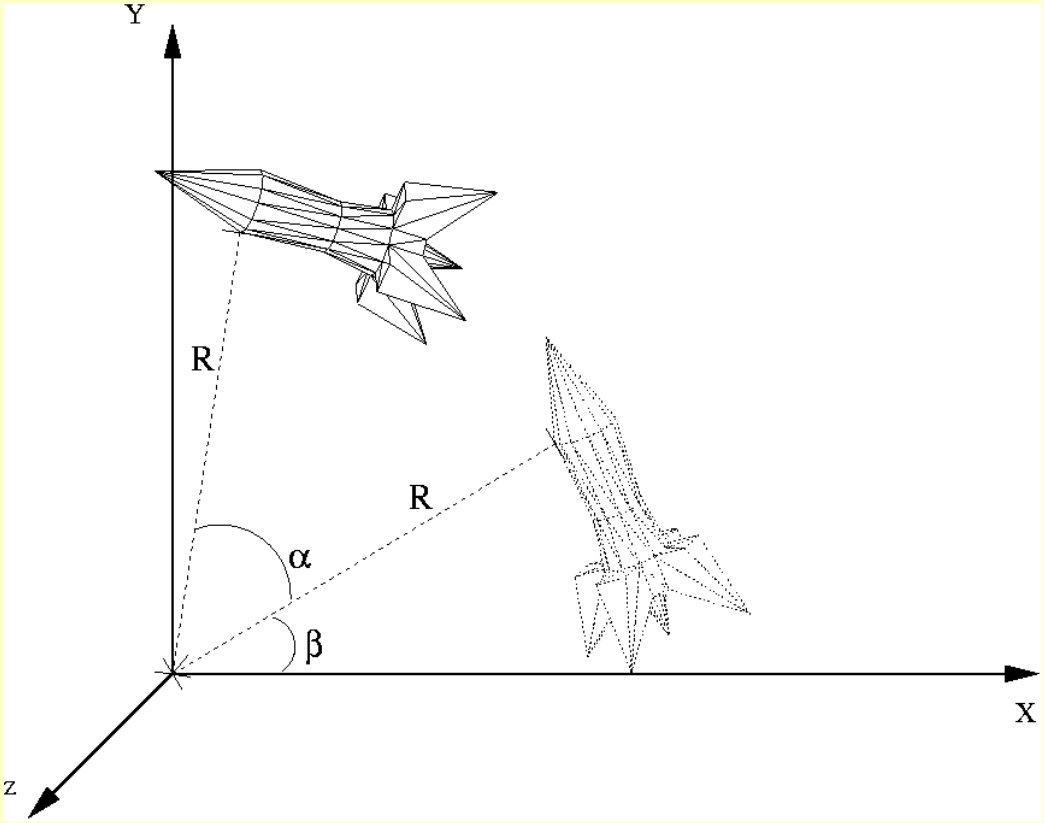
$-s_x$



- $x = R \cdot \cos(b)$
- $y = R \cdot \sin(b)$
- $z = z$
- $x' = R \cdot \cos(a+b)$
- $x' = R \cdot \cos(b) \cdot \cos(a) - R \cdot \sin(b) \cdot \sin(a) = x \cdot \cos(a) - y \cdot \sin(a)$
- $y' = R \cdot \sin(a+b)$
- $y' = R \cdot \cos(b) \cdot \sin(a) + R \cdot \sin(b) \cdot \cos(a) = x \cdot \sin(a) + y \cdot \cos(a)$

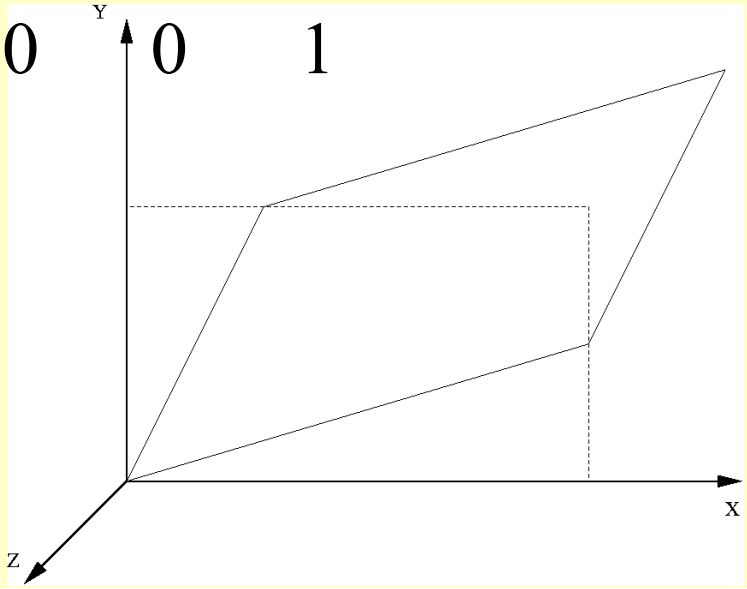
cos (a)	-sin (a)	0	0
sin (a)	cos (a)	0	0
0	0	1	0
0	0	0	1

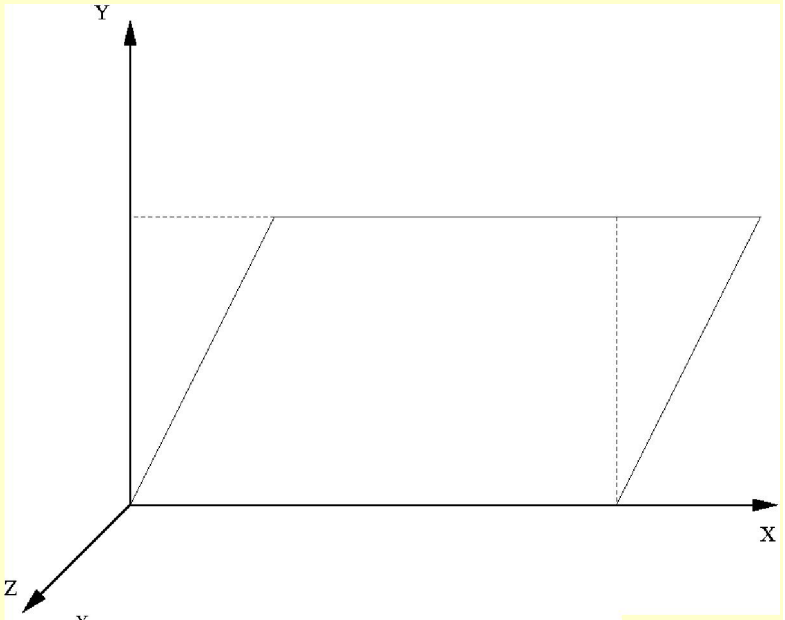
- Around z axis
- cos (a) -sin (a) 0 0
- sin (a) cos (a) 0 0
- 0 0 1 0
- 0 0 0 1
- Around x axis
- 1 0 0 0
- 0 cos (a) -sin (a) 0
- 0 sin (a) cos (a) 0
- 0 0 1 1
- Around y axis
- cos (a) 0 -sin (a) 0
- 0 1 0 0
- sin (a) 0 cos (a) 0
- 0 0 0 1
-

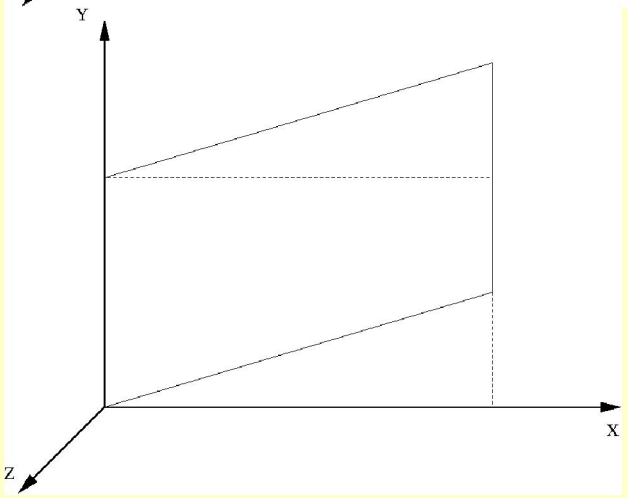


- Shear
- $x' = x+y\cdot b+z\cdot c$
- $y' = e\cdot x+y+g\cdot z$
- $z' = i\cdot x+j\cdot y+z$

- $1 \quad b \quad c \quad 0$
- $e \quad 1 \quad g \quad 0$
- $i \quad j \quad 1 \quad 0$
- $0 \quad 0 \quad 0 \quad 1$
-







Object Transformations in OpenGL

- Multiplies the current matrix by a matrix that moves/rotates an object
- glTranslate (x, y, z):
- glRotate (angle, x, y, z):counterclockwise direction in degrees.
- glScale (x, y, z): by the corresponding argument x, y, or z.
- glTranslatef(0.0, 0.0, -5.0). 5 units of distance between the viewpoint and the objects by moving the viewpoint
- gluLookAt

- OpenGL is a graphics API, the computational model behind this API, called the Graphics Pipeline Model, used for producing images of geometric models. The API provides geometric primitives and programming constructs for defining the models. The **model** is then **rendered** via a sequence of **pipeline stages:**

- Subject these **models** to several **3D transformations**,
- Followed **by clipping**, then
- **Projection to the plane**.
- Followed by **planar (2D) transformations**, and
- Finally **rasterization**: **converting to pixels**.

- World (or Scene) Coordinates

- Window Coordinates

- View (or Eye or Camera) Coordinates

- Projection Coordinates

- OpenGL primitive number type. similar types in C/C++. GLint, GLfloat and GLdouble:
- Vertices: the atoms for geometric modeling. Vertex is the most basic geometric object. This is a point, in 2 or 3d, the types of the coordinates might be int or float.
- Specifying in a program..
- Vertex(int x, int y); // 2-dimensional
- Vertex(float x, float y, float z); // 3-dimensional
- Specifying in gl, (OpenGl syntax)
- glVertex2i(GLint x, GLint y);
- glVertex3f(GLfloat x, GLfloat y, GLfloat z);
-

Object Transformations in OpenGL

- The transformation process to produce the desired scene for viewing is analogous to taking picture with a camera.
- The steps with a camera (or a computer):

A. **Modeling transformation**: Arranging the scene to be photographed into the desired composition.

B. **Viewing transformation**: Setting up your tripod and pointing the camera at the scene .

C. **Projection transformation**: Choosing a camera lens or adjusting the zoom.

D. **Viewport transformation**:Determining how large you want the final photograph to be for example, you might want it enlarged.

Not necessarily the order in your code should follow this order, but the **viewing transformations must precede the modeling transformations in your code**, but you can specify the projection and viewport transformations at any point before drawing occurs.

The order in which these operations occur on your computer ???.

Construct a 4×4 matrix M to specify viewing, modeling, and projection transformations, which is then multiplied by the coordinates of each vertex v.

v'=Mv

Note that **viewing and modeling transformations are automatically applied to surface normal vectors to form modelview matrix, which is applied to the incoming object coordinates to yield eye coordinates,** , (Normal vectors are used only in eye coordinates.) This ensures that the normal vector's relationship to the vertex data is properly preserved. Next, if you've specified arbitrary clipping planes to remove certain objects from the scene or to provide cutaway views of objects, these clipping planes are applied.