When we have got set of three dimensional points with coordinates X, Y and Z we can rotate it using usual equations for rotation around x, y and z axis separately. We can also transform three rotations by multiplication of it's matrices and rotate it at one take:

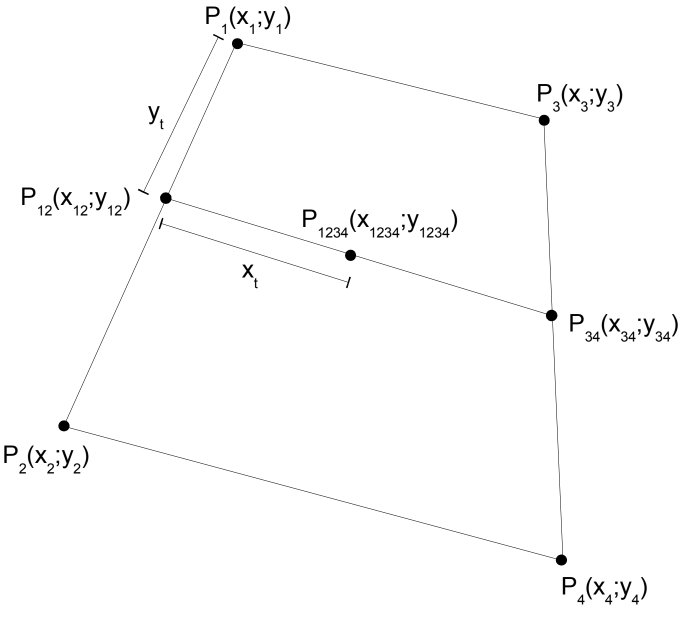
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \times \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \times \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) & \sin(\beta) \\ \sin(\alpha)\sin(\beta)\cos(\gamma)+\cos(\alpha)\sin(\gamma) & -\sin(\alpha)\sin(\beta)\cos(\gamma)+\cos(\alpha)\cos(\gamma) & -\sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta)\cos(\gamma)+\sin(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma)+\sin(\alpha)\cos(\gamma) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

$$X_R = X\cos(\beta)\cos(\gamma) - Y\cos(\beta)\sin(\gamma) + Z\sin(\beta)$$
$$Y_R = X(\sin(\alpha)\sin(\beta)\cos(\gamma)+\cos(\alpha)\sin(\gamma)) + Y(-\sin(\alpha)\sin(\beta)\cos(\gamma)+\cos(\alpha)\cos(\gamma)) - Z\sin(\alpha)\cos(\beta)$$
$$Z_R = X(-\cos(\alpha)\sin(\beta)\cos(\gamma)+\sin(\alpha)\sin(\gamma)) + Y(\cos(\alpha)\sin(\beta)\sin(\gamma)+\sin(\alpha)\cos(\gamma)) + Z\cos(\alpha)\cos(\beta)$$

Last three lines are equations of point coordinates after complex rotation around three axis: x, y and z given by three angles: α, β and γ.
Since we've got points coordinates after rotation, we move the set of points along z axis by c.a. 400 units, and transform by equations of perspective. We will have set of two dimensional coordinates of points. For each surface we define four corners and we align a texture along it. There are four points named $P_1(x_1;y_1)$, $P_2(x_2;y_2)$, $P_3(x_3;y_3)$ and $P_4(x_4;y_4)$, with coordinates x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4.
Another point $P_{12}(x_{12},y_{12})$ moves on the way from point P_1 to point P_2. It moves by linear interpolation of it's coordinates between point P_1 and P_2:

$$x_{12}=x_1+y_t(x_2-x_1)$$
$$y_{12}=y_1+y_t(y_2-y_1)$$



Exactly the same way moves point $P_{34}(x_{34};y_{34})$:

$$x_{34}=x_3+y_t(x_4-x_3)$$
$$y_{34}=y_3+y_t(y_4-y_3)$$

x_t and y_t are indexes of proportion of width and height of a texture. Indexes range from 0 to 1. 0 is left edge of a texture for index x_t, 1 is right edge, and 0 is top edge for y_t index and 1 for bottom edge. To scale it to a full size of texture multiplicate it by dimension of a texture. $1920x_t$ and $1080y_t$ are example for a texture of high definition format. Such multiplicated x_t and y_t are pixel coordinate on a texture, which has to be drawn in a position of a point $P_{1234}(x_{1234};y_{1234})$ on a screen, which we drop our texture on. Point P_{1234} is linear interpolation of a position between previously interpolated points P_{12} and P_{34}. So, it's coordinates goes:

$$x_{1234}=x_{12}+x_t(x_{34}-x_{12})$$
$$y_{1234}=y_{12}+x_t(y_{34}-y_{12})$$

which in basics is:

$$x_{1234}=x_1+y_t(x_2-x_1)+x_t[(x_3+y_t(x_4-x_3))-(x_1+y_t(x_2-x_1))]$$
$$y_{1234}=y_1+y_t(y_2-y_1)+x_t[(y_3+y_t(y_4-y_3))-(y_1+y_t(y_2-y_1))]$$

$$x_{1234}=x_1+y_tx_2-y_tx_1+x_t[(x_3+y_tx_4-y_tx_3)-(x_1+y_tx_2-y_tx_1)]$$
$$y_{1234}=y_1+y_ty_2-y_ty_1+x_t[(y_3+y_ty_4-y_ty_3)-(y_1+y_ty_2-y_ty_1)]$$

$$x_{1234}=x_1+y_tx_2-y_tx_1+x_t(x_3+y_tx_4-y_tx_3-x_1-y_tx_2+y_tx_1)$$
$$y_{1234}=y_1+y_ty_2-y_ty_1+x_t(y_3+y_ty_4-y_ty_3-y_1-y_ty_2+y_ty_1)$$

$$x_{1234}=x_1+y_tx_2-y_tx_1+x_tx_3+x_ty_tx_4-x_ty_tx_3-x_tx_1-x_ty_tx_2+x_ty_tx_1$$
$$y_{1234}=y_1+y_ty_2-y_ty_1+x_ty_3+x_t y_ty_4-x_t y_ty_3-x_t y_1-x_t y_ty_2+x_t y_ty_1$$

It would be easy, when we would get position on a texture (coordinates x_t and y_t) having point on screen P_{1234} given. In such a case we wouldn't redraw some points of a texture, that overlap on the same pixel on a screen after mapping. We would have either no gaps in texture mapping, that could happen if distance of neighbouring pixels on texture would lay in distance bigger than one pixel on target screen after the mapping. To do that we transform equation to get x_t and y_t on a texture, of calculations for given x_{1234} and y_{1234}, belonging to target screen pixel coordinate. Then we could scan target screen with lines and columns to get texture position for each scanned point, with no overlapping or gaps.

We sort our equation for x_t, x_ty_t and y_t:

$$x_t(x_3-x_1)+x_ty_t(x_4-x_3-x_2+x_1)+y_t(x_2-x_1)=x_{1234}-x_1$$
$$x_t(y_3-y_1)+x_ty_t(y_4-y_3-y_2+y_1)+y_t(y_2-y_1)=y_{1234}-y_1$$

let it happen, that:

$$a_1=x_3-x_1$$
$$b_1=x_4-x_3-x_2+x_1$$
$$c_1=x_2-x_1$$
$$d_1=x_{1234}-x_1$$

$$a_2=y_3-y_1$$
$$b_2=y_4-y_3-y_2+y_1$$
$$c_2=y_2-y_1$$
$$d_2=y_{1234}-y_1$$

Now, the set of our two equations simplifies:

$$x_ta_1+x_ty_tb_1+y_tc_1=d_1$$
$$x_ta_2+x_ty_tb_2+y_tc_2=d_2$$

to reduce number of variables we transform the first equation:

$$x_t(a_1+y_tb_1)+y_tc_1=d_1$$
$$x_t=\frac{d_1-y_tc_1}{a_1+y_tb_1}$$

and by substituting x_t in second formula we're getting:

$$\frac{d_1-y_tc_1}{a_1+y_tb_1}a_2+\frac{d_1-y_tc_1}{a_1+y_tb_1}y_tb_2+y_tc_2=d_2$$

$$\frac{d_1a_2-y_tc_1a_2}{a_1+y_tb_1}+\frac{d_1y_tb_2-y_t^2c_1b_2}{a_1+y_tb_1}+\frac{a_1y_tc_2+y_t^2b_1c_2}{a_1+y_tb_1}=d_2$$

$$\frac{d_1a_2-y_tc_1a_2+d_1y_tb_2-y_t^2c_1b_2+a_1y_tc_2+y_t^2b_1c_2}{a_1+y_tb_1}=d_2$$

$$d_1a_2-y_tc_1a_2+d_1y_tb_2-y_t^2c_1b_2+a_1y_tc_2+y_t^2b_1c_2=d_2(a_1+y_tb_1)$$

$$d_1a_2-y_tc_1a_2+d_1y_tb_2-y_t^2c_1b_2+a_1y_tc_2+y_t^2b_1c_2=d_2a_1+y_tb_1d_2$$

later, by sorting powers of y_t, we get simple quadratic equation:

$$y_t^2(b_1c_2-c_1b_2)+y_t(a_1c_2-c_1a_2+d_1b_2-b_1d_2)+(d_1a_2-d_2a_1)=0$$

of which we calculate delta and $y_t\in<0;1>$.

$$\Delta=(a_1c_2-c_1a_2+d_1b_2-b_1d_2)^2-4(b_1c_2-c_1b_2)(d_1a_2-d_2a_1)$$

usually it's the root with substracted delta, that falls in desired range:

$$y_t=\frac{-a_1c_2+c_1a_2-d_1b_2+b_1d_2-\sqrt{\Delta}}{2(b_1c_2-c_1b_2)}$$

if $b_1c_2-c_1b_2=0$, or y_t is given with equation:

$$y_t=\frac{d_2a_1-d_1a_2}{a_1c_2-c_1a_2+d_1b_2-b_1d_2}$$

If delta is less than zero we don't draw the point, and when y_t is outside th <0;1> range we don't draw it either.

Calculating x_t of y_t may bring technical problems of division by zero in unexpected moments, so it is safer to proceed with above elaborate, transforming the second equation:

$$x_ta_2+x_ty_tb_2+y_tc_2=d_2$$

into y_t:

$$y_t=\frac{d_2-x_ta_2}{c_2+x_tb_2}$$

when inserting into:

$$x_ta_1+x_ty_tb_1+y_tc_1=d_1$$

we get:

$$x_ta_1+x_t\frac{d_2-x_ta_2}{c_2+x_tb_2}b_1+\frac{d_2-x_ta_2}{c_2+x_tb_2}c_1=d_1$$

and after transformation and reduction:

$$\frac{x_ta_1c_2+x_t^2a_1b_2}{c_2+x_tb_2}+\frac{x_td_2b_1-x_t^2a_2b_1}{c_2+x_tb_2}+\frac{d_2c_1-x_ta_2c_1}{c_2+x_tb_2}=d_1$$

$$\frac{x_ta_1c_2+x_t^2a_1b_2+x_td_2b_1-x_t^2a_2b_1+d_2c_1-x_ta_2c_1}{c_2+x_tb_2}=d_1$$

$$x_ta_1c_2+x_t^2a_1b_2+x_td_2b_1-x_t^2a_2b_1+d_2c_1-x_ta_2c_1=d_1(c_2+x_tb_2)$$

$$x_ta_1c_2+x_t^2a_1b_2+x_td_2b_1-x_t^2a_2b_1+d_2c_1-x_ta_2c_1=d_1c_2+x_td_1b_2$$

$$x_ta_1c_2+x_t^2a_1b_2+x_td_2b_1-x_t^2a_2b_1+d_2c_1-x_ta_2c_1-d_1c_2-x_td_1b_2=0$$

$$x_t^2(a_1b_2-a_2b_1)+x_t(a_1c_2-a_2c_1+d_2b_1-d_1b_2)+(d_2c_1-d_1c_2)=0$$

of which we calculate delta and $x_t\in$<0;1>:

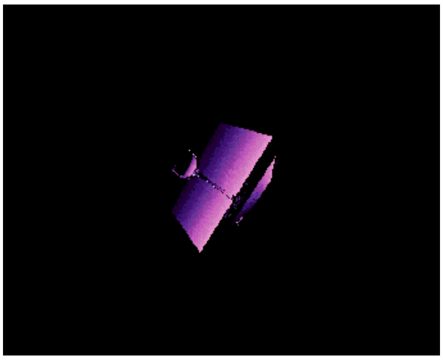
$$\Delta=(a_1c_2-c_1a_2+d_1b_2-b_1d_2)^2-4(a_1b_2-b_1a_2)(d_2c_1-d_1c_2)$$

$$x_t=\frac{-a_1c_2+c_1a_2-d_1b_2+b_1d_2-\sqrt{\Delta}}{2(a_1b_2-a_2b_1)}$$

If $a_1b_2-b_1a_2=0$, we use formula:

$$x_t=\frac{d_1c_2-d_2c_1}{a_1c_2-c_1a_2+d_1b_2-b_1d_2}$$

Since we've got x_t and y_t corresponding to x_{1234} and y_{1234} on target screen, we can draw a point in same colour that is the colour of a pixel x_t, y_t on texture, obviously after scaling x_t and y_t to dimensions of a texture by multiplication by texture's width and height.



Sharky's picture named Kieliszek from GFX compo on Polish Autumn 1993.