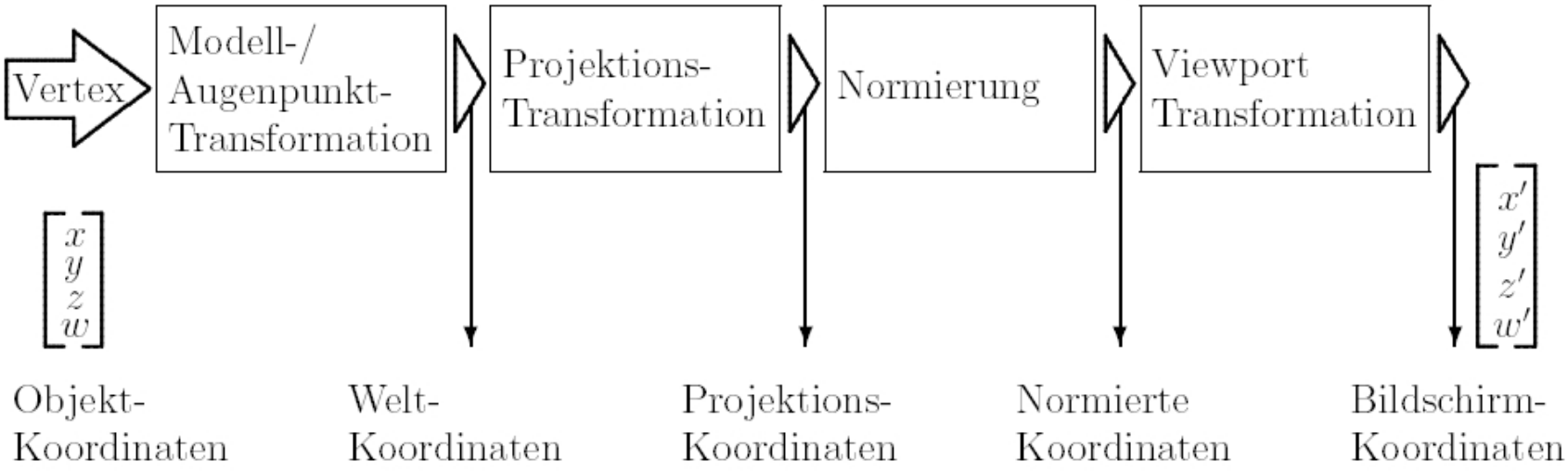
Vertex-Transformations-Stufen

glTranslate
glRotate
glScale

gluLookAt

glFrustum
gluPerspective
glOrtho

glViewport



(Nischwitz *et al.*, 2011)

Transformations-Matrizen (1)

- **glMatrixMode**, mit Argument: GL_MODELVIEW (Vorgabewert), GL_PROJECTION, GL_TEXTURE
- **glLoadIdentity** - ersetzt die aktuelle Matrix durch die Einheitsmatrix
- **glLoadMatrix{fd}**(TYPE[] *m*, int *m_offset*) - lädt die spezifizierte Matrix *m* als die aktuell anzuwendende Matrix
- **glMultMatrix{fd}**(TYPE[] *m*, int *m_offset*) - multipliziert die spezifizierte Matrix *m* mit der aktuell im Speicher befindlichen Matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translations-Matrix

$$m = \{m1, m2, m3, m4, \ldots, m16\}$$

$$M = \begin{bmatrix} m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \\ m4 & m8 & m12 & m16 \end{bmatrix}$$

```
float m[] = {
  1.0f, 0.0f, 0.0f, 0.0f,
  0.0f, 1.0f, 0.0f, 0.0f,
  0.0f, 0.0f, 1.0f, 0.0f,
 -3.0f, 0.0f, 0.0f, 1.0f};

gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glMultMatrixf(m, 0);

// gl.glTranslatef(-3.0f, 0.0f, 0.0f);
```

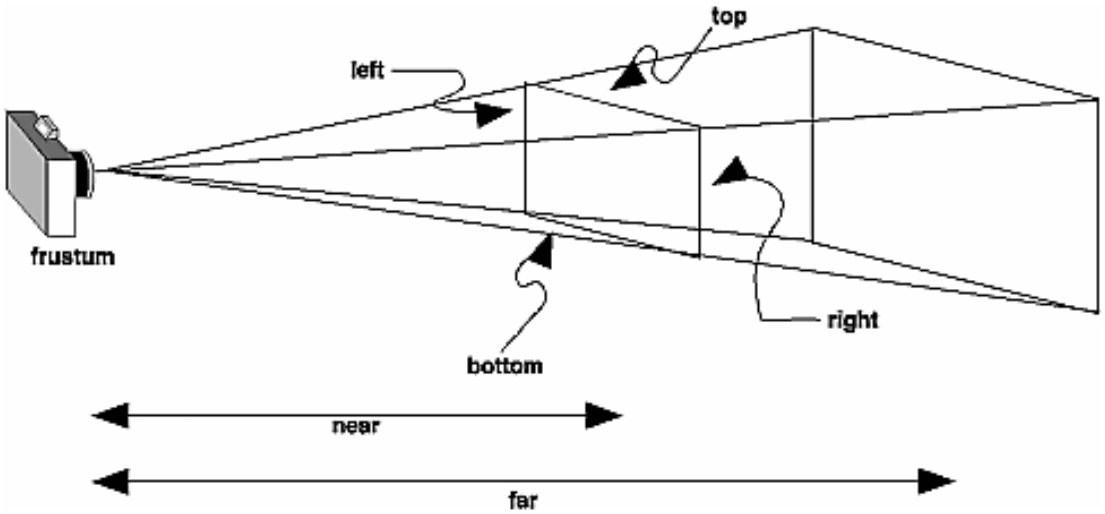
Transformations-Matrizen (2)

- **glGetFloatv**(int *pname*, float *params*, int *params_offset*)
  glGetDoublev(int *pname*, double *params*, int *params_offset*)

```
float m[] = new float[16];

// return the modelview matrix (on the top of the modelview matrix stack)
gl.glGetFloatv(GL.GL_MODELVIEW_MATRIX, m, 0);

// return the projection matrix (on the top of the projection matrix stack)
gl.glGetFloatv(GL.GL_PROJECTION_MATRIX, m, 0);

// return the projection matrix (on the top of the texture matrix stack)
gl.glGetFloatv(GL.GL_TEXTURE_MATRIX, m, 0);
```

Perspektivische Projektion (1)

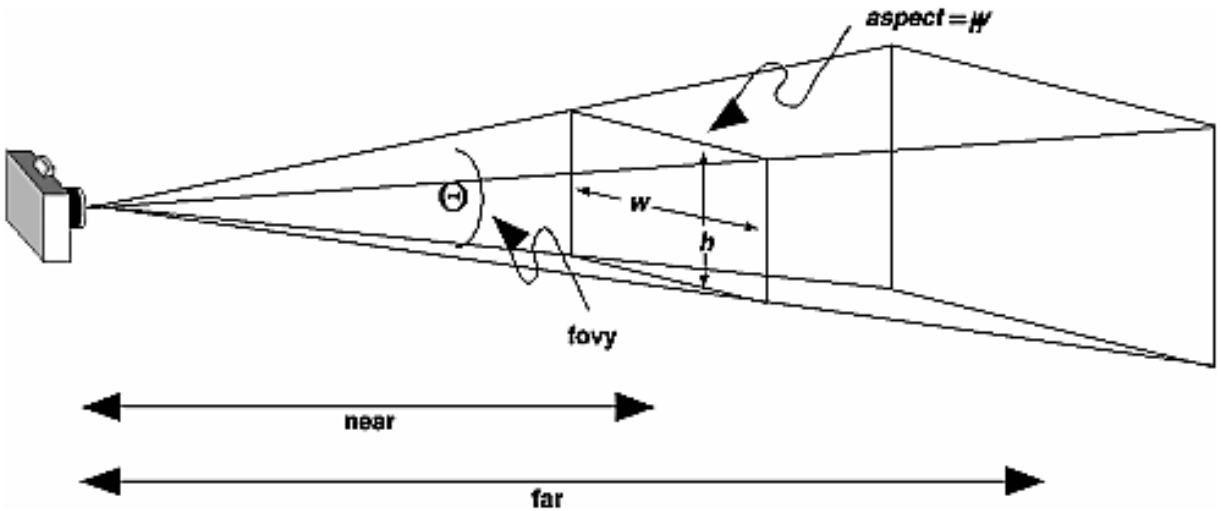
- **glFrustum**(double *left*, double *right*, double *bottom*, double *top*, double *near*, double *far*)



`http://www.glprogramming.com/red/chapter03.html`

Perspektivische Projektion (2)

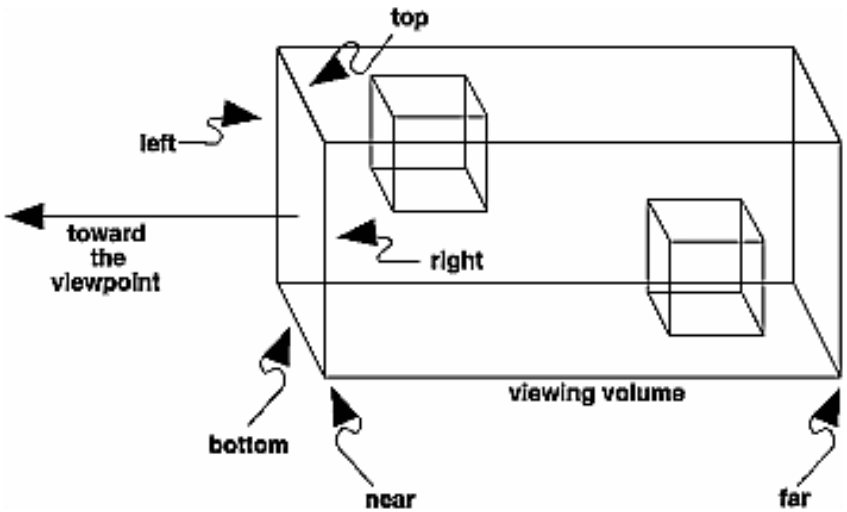
- **gluPerspective**(double *fovy*, double *aspect*, double *near*, double *far*)



http://www.glprogramming.com/red/chapter03.html

Orthographische Projektion

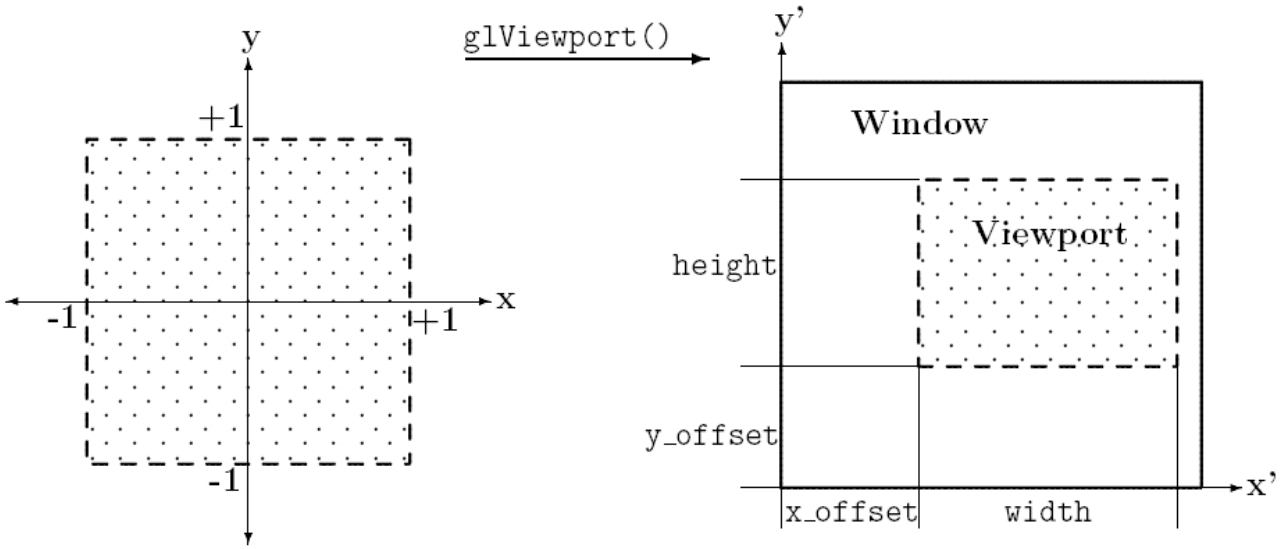
- **glOrtho**(double *left*, double *right*, double *bottom*, double *top*, double *near*, double *far*)



http://www.glprogramming.com/red/chapter03.html

Viewport-Transformation (1)

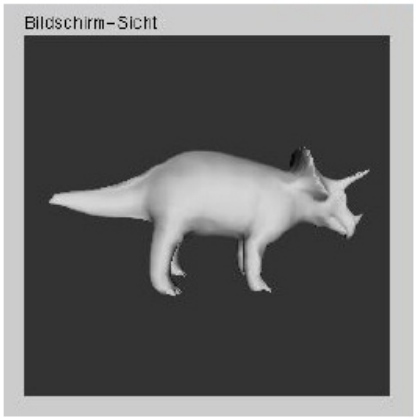
- Umrechnung von normierten Koordinaten in Bildschirm-Koordinaten
- **glViewport**(int *x*, int *y*, int *width*, int *height*)



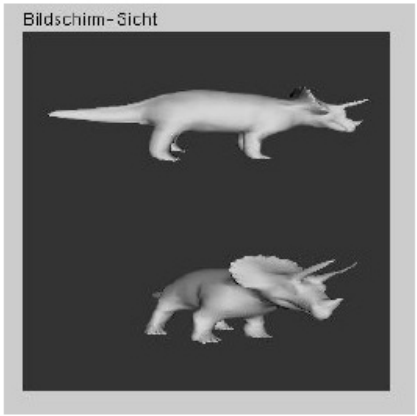
(Nischwitz *et al.*, 2011)

Viewport-Transformation (2)

- Ein Bildschirm-Fenster kann mehrere Viewports enthalten



`glViewport(0,0,256,256)`



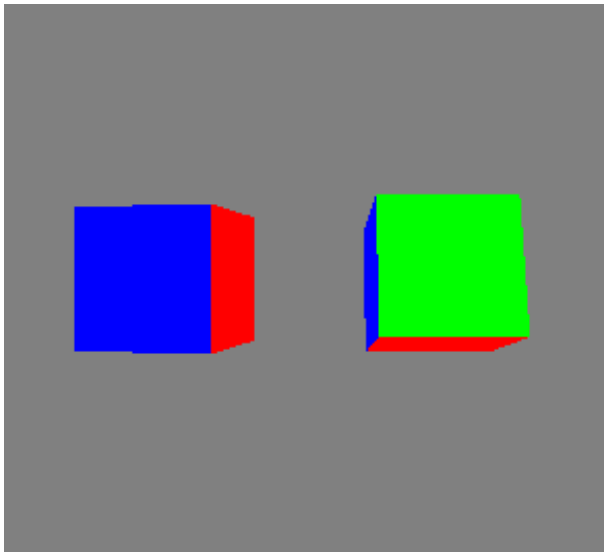
`glViewport(0,128,256,128)`
`glViewport(0,0,256,128)`

(Nischwitz *et al.*, 2011)

Matrizen-Stapel

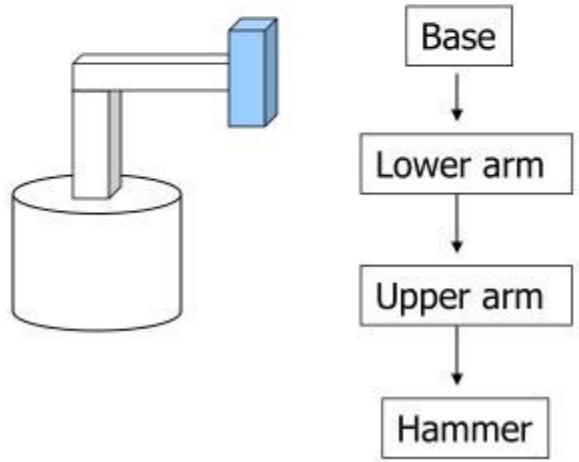
- Für hierarchisch aufgebaute Objekte
- Für jede Hierarchiestufe wird eine eigene Gesamttransformations-Matrix in einem Stapel gespeichert
- **glPushMatrix**, **glPopMatrix**

```
public void display(GLAutoDrawable drawable) {
    ....
    gl.glLoadIdentity(); gl.glTranslatef(0.0f,
    0.0f, -10.0f);

    // save matrix state
    gl.glPushMatrix();
        gl.glTranslatef(-2.0f, 0.0f, 0.0f);
        gl.glRotatef(angle, 0.0f, 1.0f, 0.0f);
        drawCube(gl);
    // restore matrix state
    gl.glPopMatrix();
    gl.glTranslatef(2.0f, 0.0f, 0.0f);
    gl.glRotatef(angle, 1.0f, 0.0f, 0.0f);
    drawCube(gl);
}
```



Do it in OpenGL

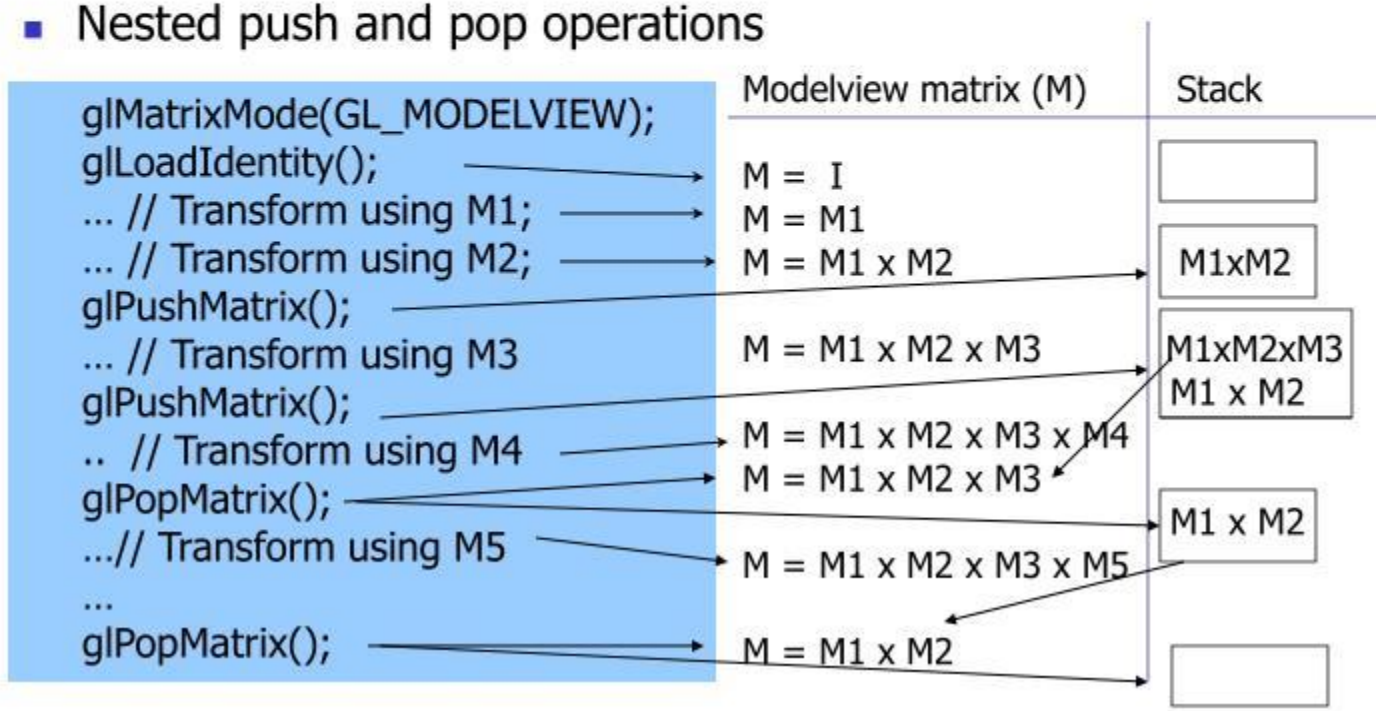
- Translate base and all its descendants by (5,0,0)
- Rotate the lower arm and its descendants by -90 degree about the local y




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

... // setup your camera

glTranslatef(5,0,0);

Draw_base();

glRotatef(-90, 0, 1, 0);

Draw_lower _arm();
Draw_upper_arm();
Draw_hammer();
```

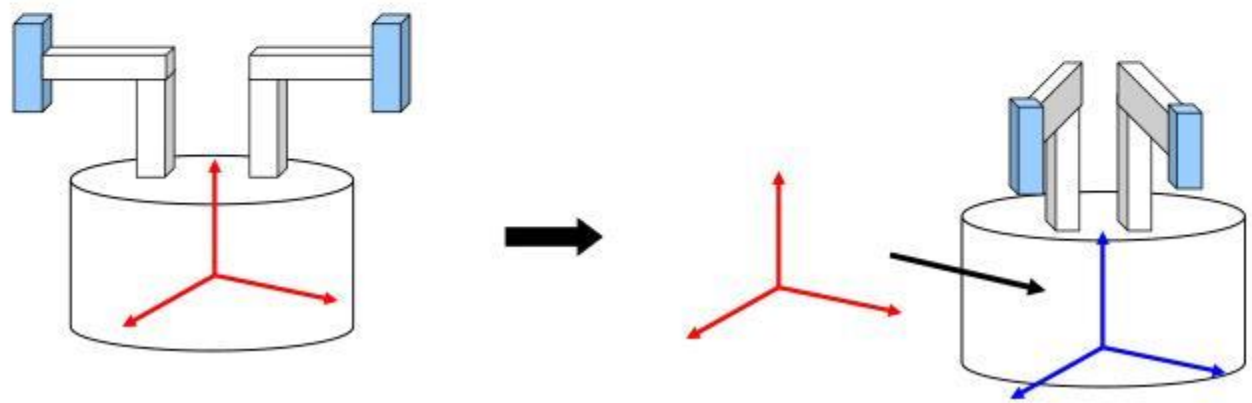
Push and Pop Matrix Stack

- Nested push and pop operations



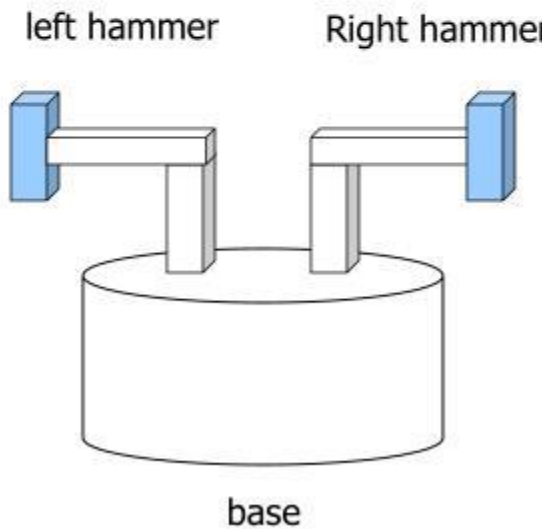
Do this …

- Base and everything – translate (5,0,0)
- Left hammer – rotate 75 degree about the local y
- Right hammer – rotate -75 degree about the local y

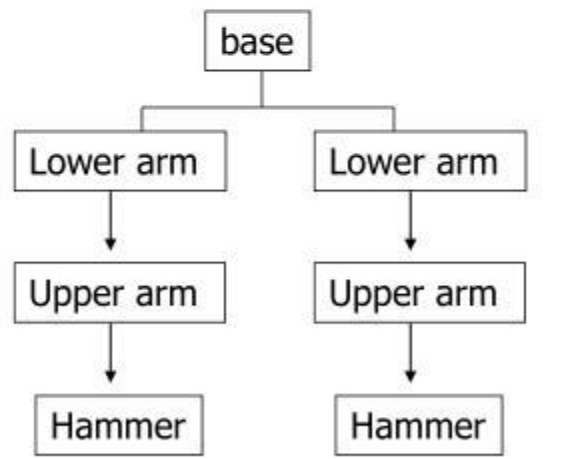


A more complicated example

- How about this model?



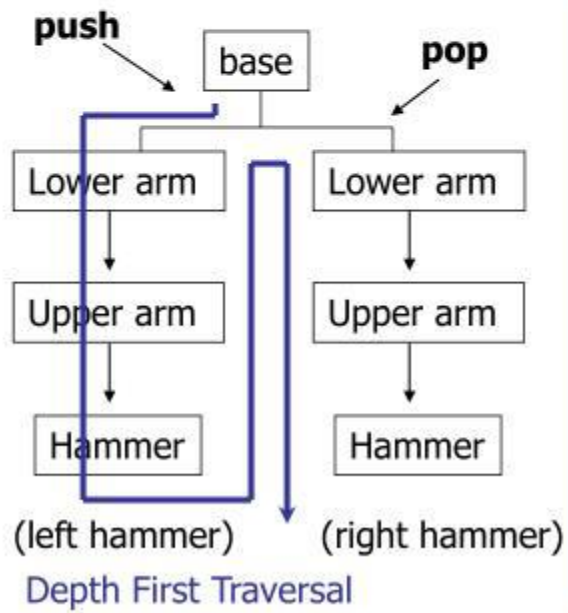
Scene Graph?



(left hammer) (right hammer)

Push and Pop Matrix Stack

- A simple OpenGL routine:




```
glTranslate(5,0,0)
Draw_base();
glPushMatrix();

glRotate(75, 0,1,0);
Draw_left_hammer();

glPopMatrix();
glRotate(-75, 0,1,0);
Draw_right_hammer();
```

4.5 Tastatur-Ereignisse verarbeiten

Unser Programm soll nun auf Tastatureingaben reagieren und einen Würfel im Ursprung aus- und wieder einblenden können. Dazu benötigen wir wie aus Java bekannt einen KeyListener. Wir erstellen dazu wiederum eine innere Klasse mit Namen „MyKeyListener“, die das Interface KeyListener implementiert.

```
private class MyKeyListener implements KeyListener {

	public void keyPressed(KeyEvent e) {
	}

	public void keyReleased(KeyEvent e) {
	}

	public void keyTyped(KeyEvent e) {
		switch (e.getKeyChar()) {
			case 0x1B:
				System.exit(0);
				break;
			case 0x20:
				showCube = !showCube;
				canvas.display();
				break;
			default:
				break;
		}
	}
}
```

Es soll folgende Eingabemöglichkeiten geben:

- Escape-Taste (Hexcode 1B): Programm beenden
- Leertaste (Hexcode 20): Würfel ein- und ausblenden

Das Ein- und Ausblenden des Würfels wird über eine globale, Boole’sche, mit „false“ initialisierte Variable „showCube“ realisiert. Dazu wird die Funktion „display“ des GLEventListeners wie folgt ergänzt und bei jedem Druck auf die Leertaste erneut aufgerufen (Die Funktion „canvas.display()“ ruft die „display“-Funktion des angehängten GLEventListeners auf):

```
if (showCube) {
	gl.glColor3f(1.0f, 1.0f, 1.0f);		// WEISS
	glut.glutWireCube(0.8f);
}
```

4.6 Maus-Ereignisse verarbeiten

Es soll nun ermöglicht werden, mit der Maus die Szene zu drehen. Dazu benötigen wir diesmal zwei Listener, nämlich einen MouseListener zur Überwachung der Tasten und einen MouseMotionListener um auf Ziehen zu testen. Sie werden genau wie der KeyListener an das GLAutoDrawable-Objekt angehängt. Ihr Code sieht so aus:

```
private class MyMouseMotionListener implements MouseMotionListener {
	public void mouseDragged(MouseEvent e) {
		int x = e.getX();
		int y = e.getY();
		viewRotY = ((x - dragStartX) / (float) getWidth())  * 360.0f;
		viewRotX = ((y - dragStartY) / (float) getHeight()) * 360.0f;
		dragStartX = e.getX();
		dragStartY = e.getY();
		canvas.display();
	}
	public void mouseMoved(MouseEvent arg0) {
	}
}

private class MyMouseListener implements MouseListener {
	public void mouseClicked(MouseEvent arg0) {
	}
	public void mouseEntered(MouseEvent arg0) {
	}
	public void mouseExited(MouseEvent arg0) {
	}
	public void mousePressed(MouseEvent e) {
		dragStartX = e.getX();
		dragStartY = e.getY();
	}
	public void mouseReleased(MouseEvent arg0) {
	}
}
```

Die Variablen viewRotX/Y (Float) und dragStartX/Y (Integer) müssen global definiert und mit 0 initialisiert werden. In der „display“-Funktion fügt man jetzt VOR dem Zeichnen der Objekte folgendes ein:

```
		gl.glRotatef(viewRotX, 1.0f, 0.0f, 0.0f);
		gl.glRotatef(viewRotY, 0.0f, 1.0f, 0.0f);
```

Damit wird die gesamte Szene gedreht und währenddessen ständig neu gezeichnet.