Chapter 6

Hierarchical Modelling

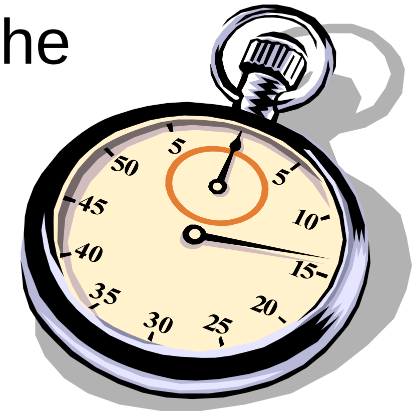
6.1 Overview

The creation and manipulation of a system representation is termed **modeling**. Any single representation is called a **model** of the system, which could be defined graphically or purely descriptively, such as a set of equations that describe the relationships between system parameters. Graphical models are often referred to as **geometric models**, because the component parts of a system are represented with geometric entities, such as straight-line segments, polygons, polyhedra, cylinders, spheres, etc.

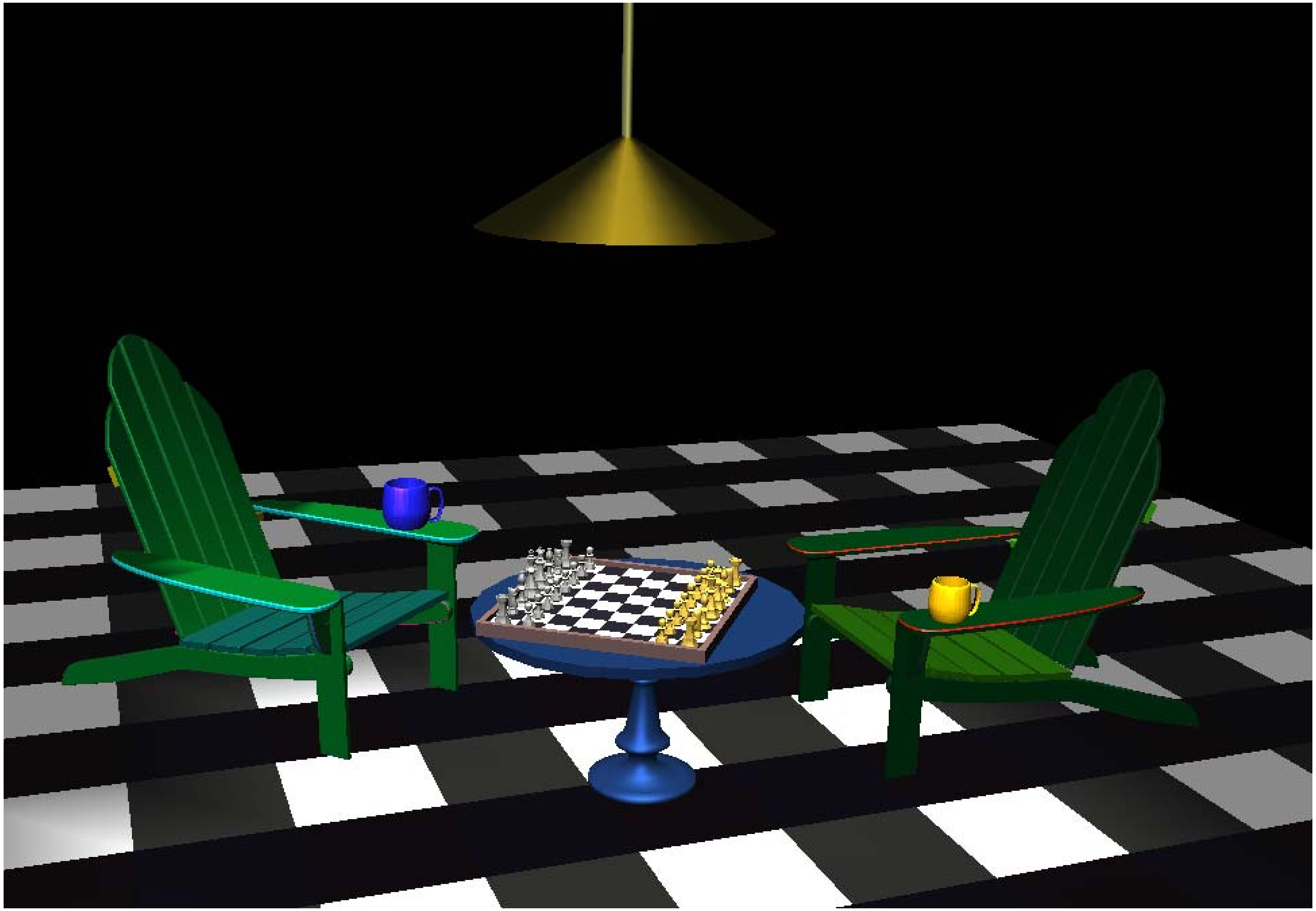
6.1 Overview

In OpenGL, triangles, quads, parametric curves and surfaces are the building blocks from which more complex real-world objects are modeled.

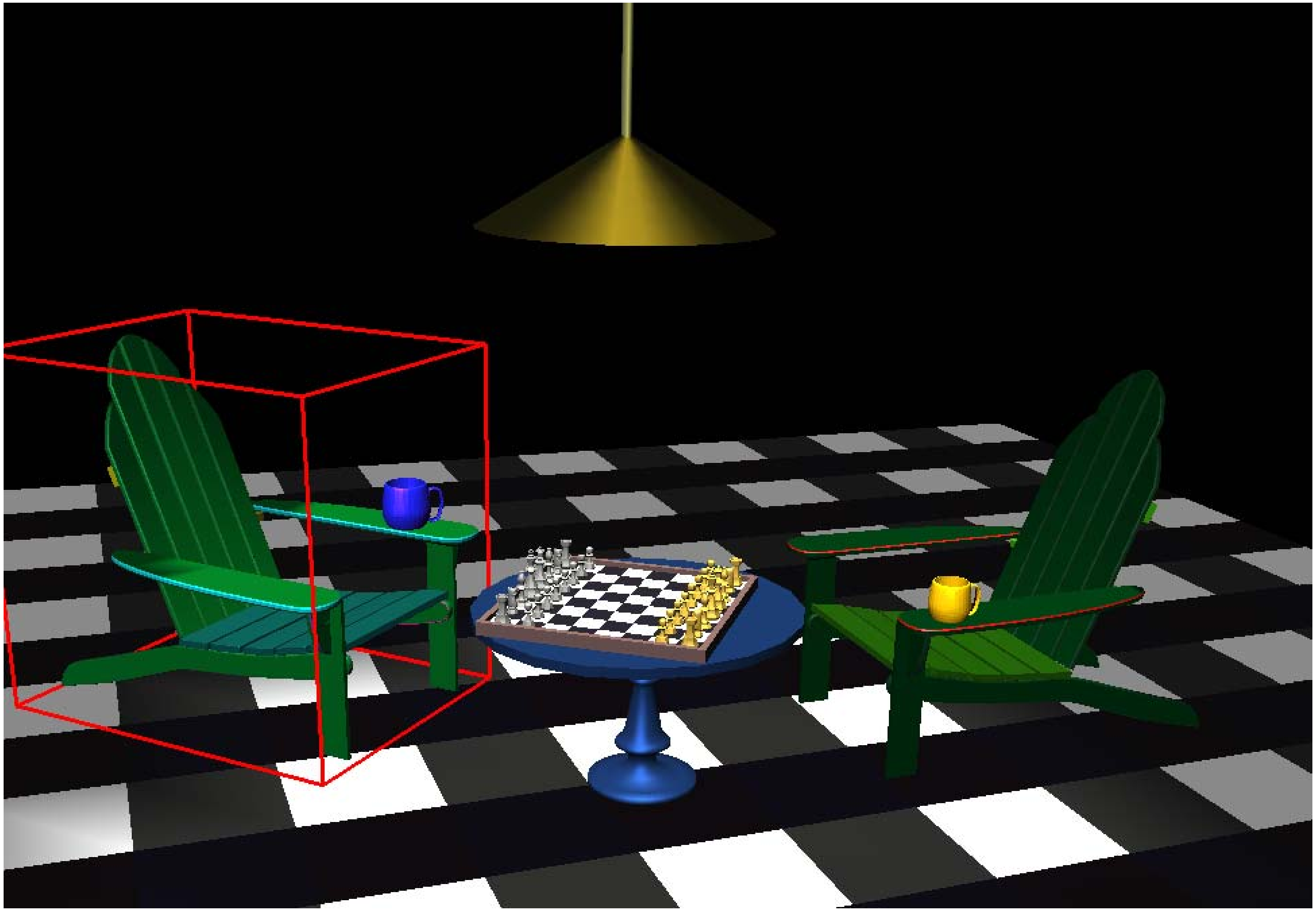
Hierarchical modeling creates complex real-world objects by combining simple primitive shapes into more complex aggregate objects while preserving the connectivity between objects.



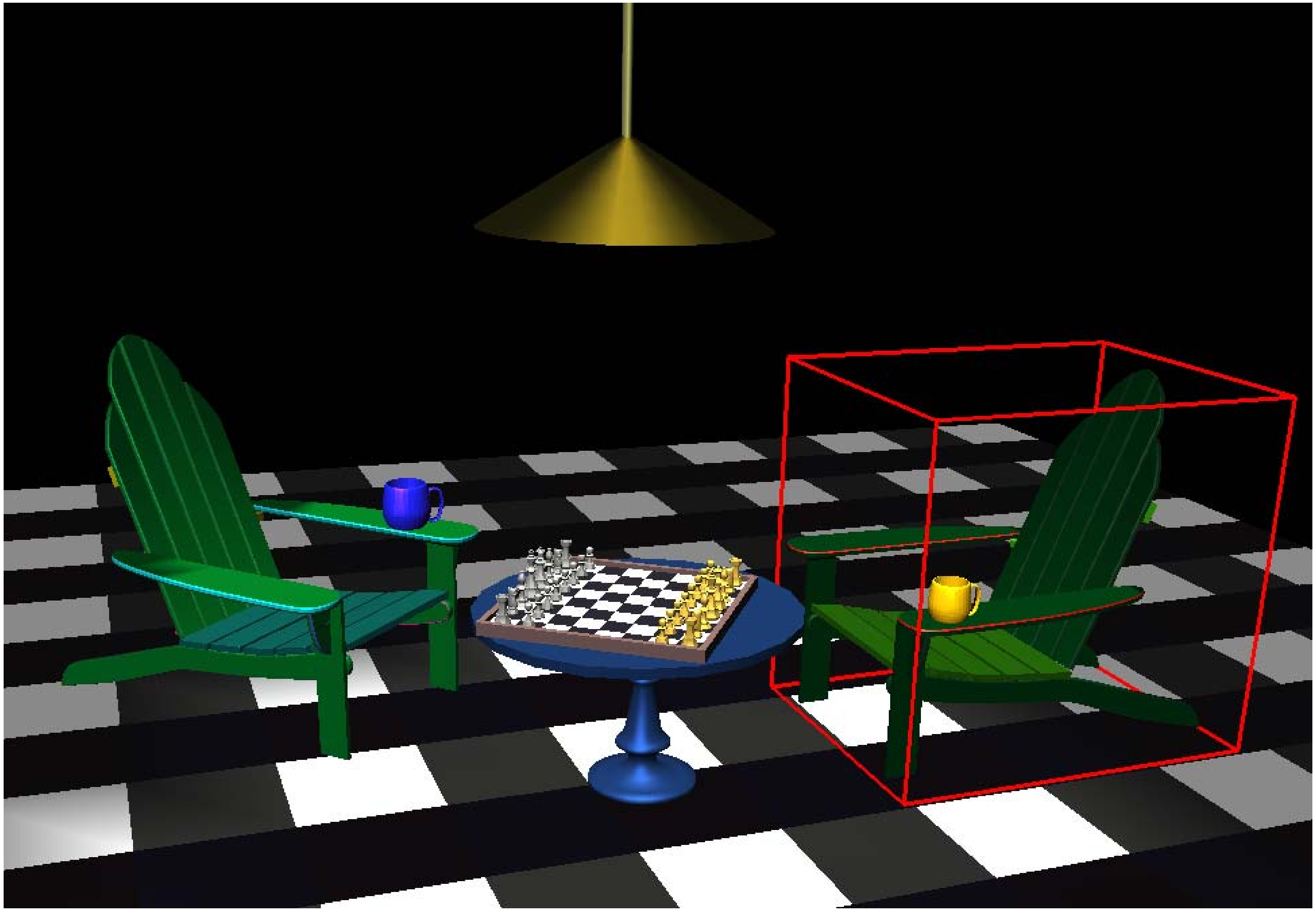
6.1 Hierarchical models



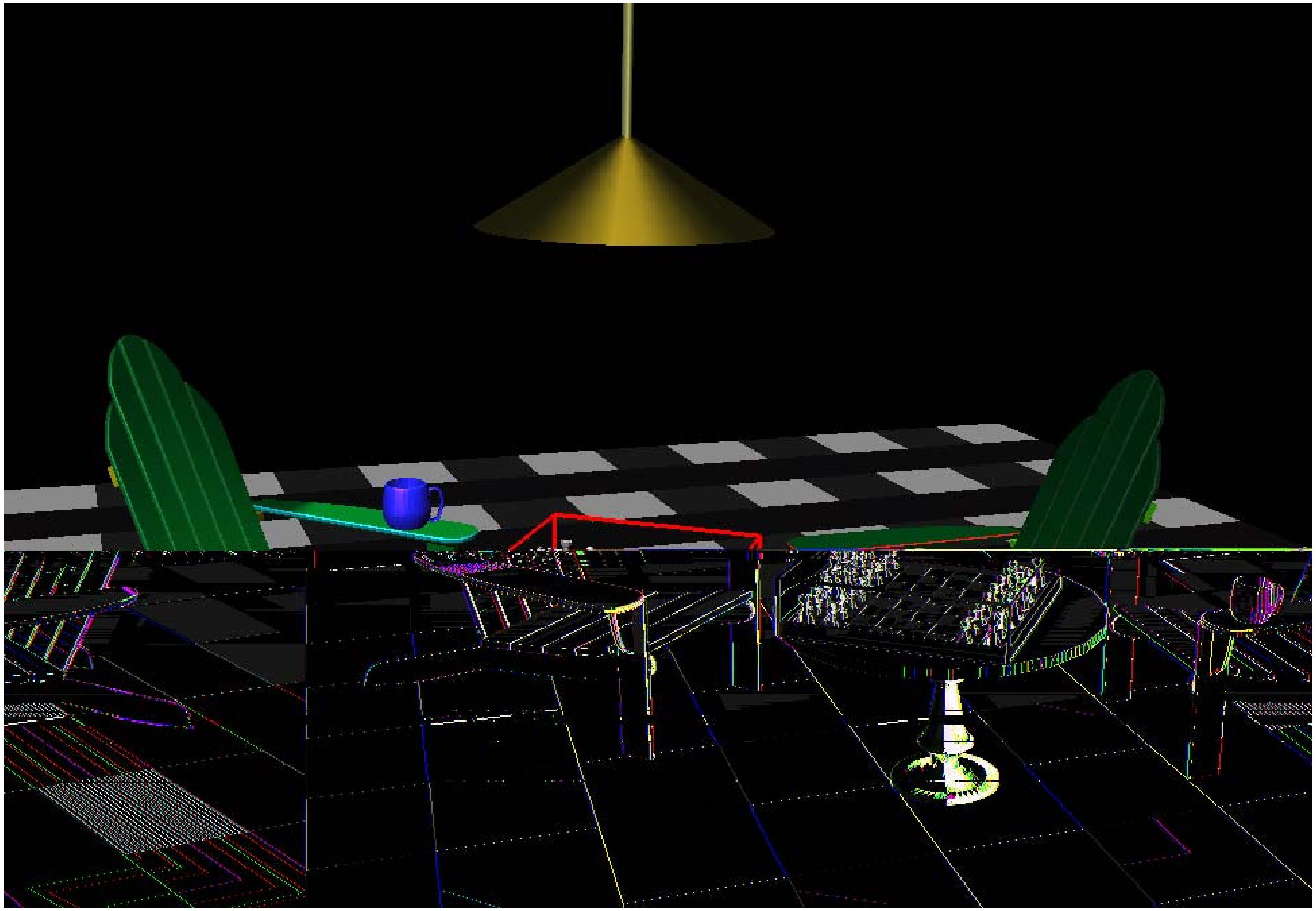
6.1 Hierarchical models



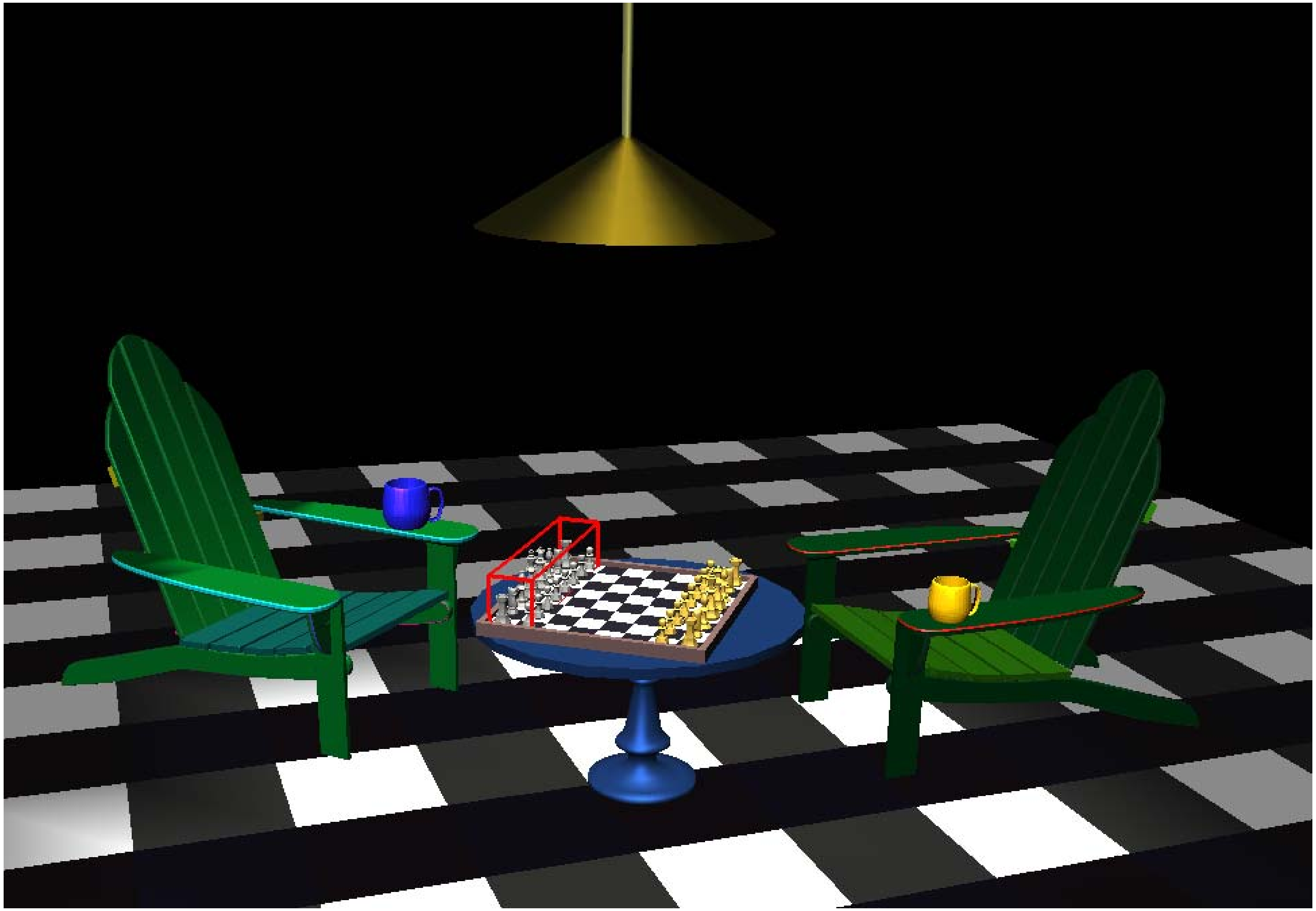
6.1 Hierarchical models



6.1 Hierarchical models



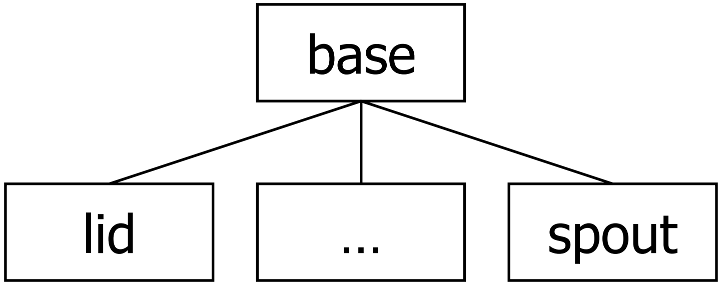
6.1 Hierarchical models

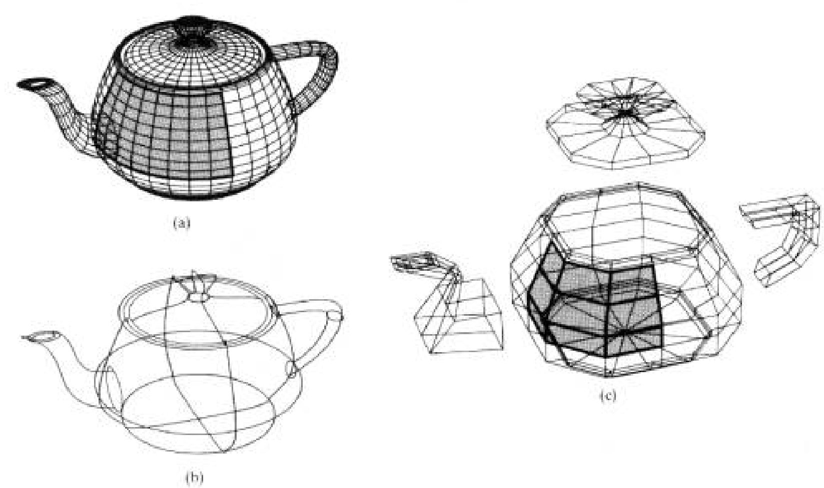


6.1 Hierarchical models

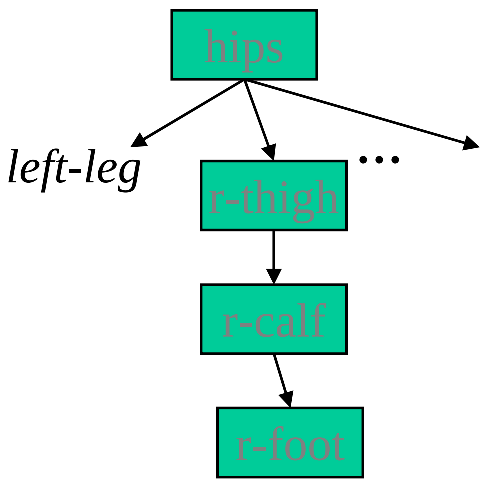
6.1 Hierarchical models

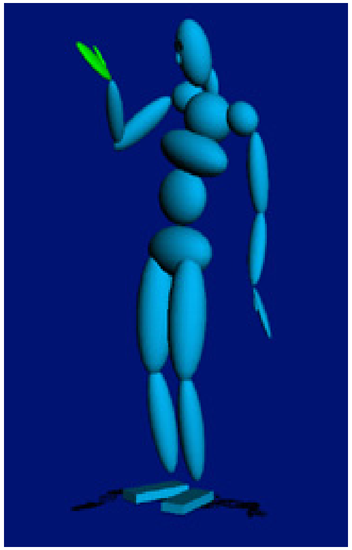
Curves and surfaces





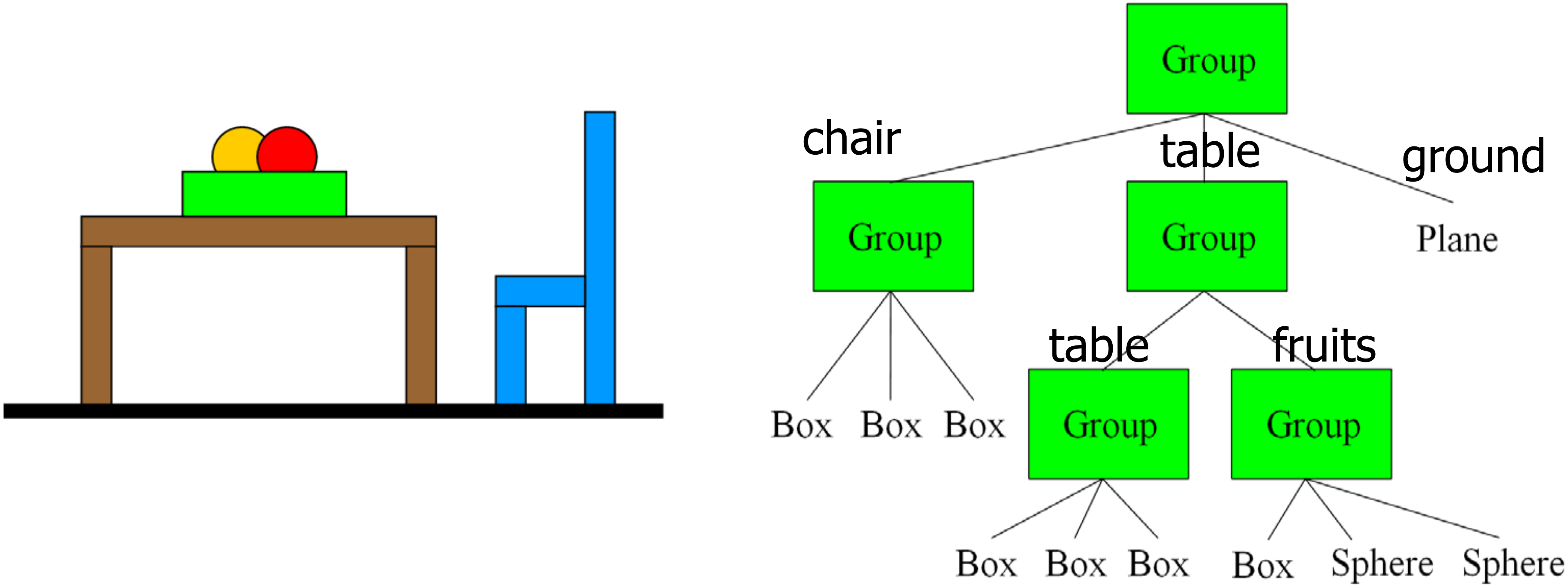
Animated Characters





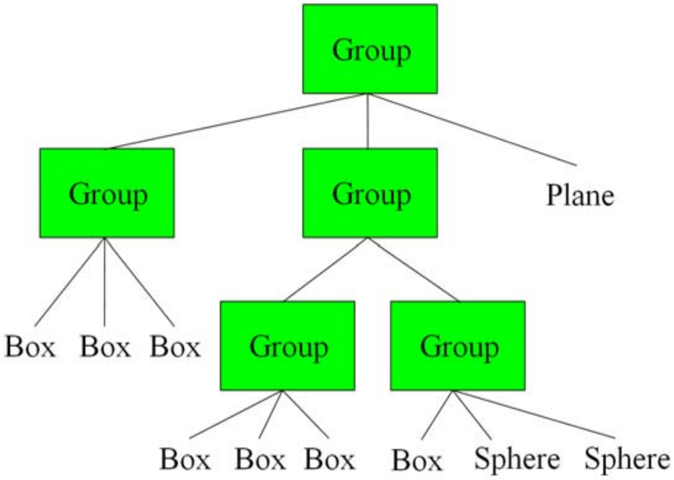
6.1 Hierarchical models

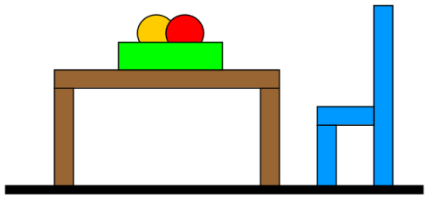
Logical organization of scene



6.1 Hierarchical models

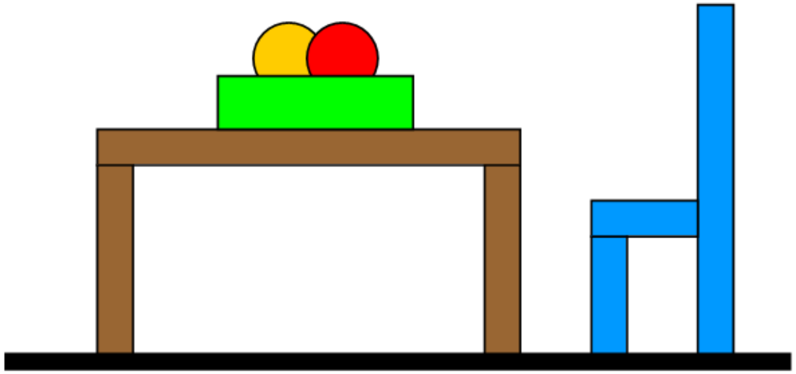
```
Group {
    numObjects 3
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Box { <BOX PARAMS> }
            Sphere { <SPHERE PARAMS> }
            Sphere { <SPHERE PARAMS> } } }
    Plane { <PLANE PARAMS> } }
```



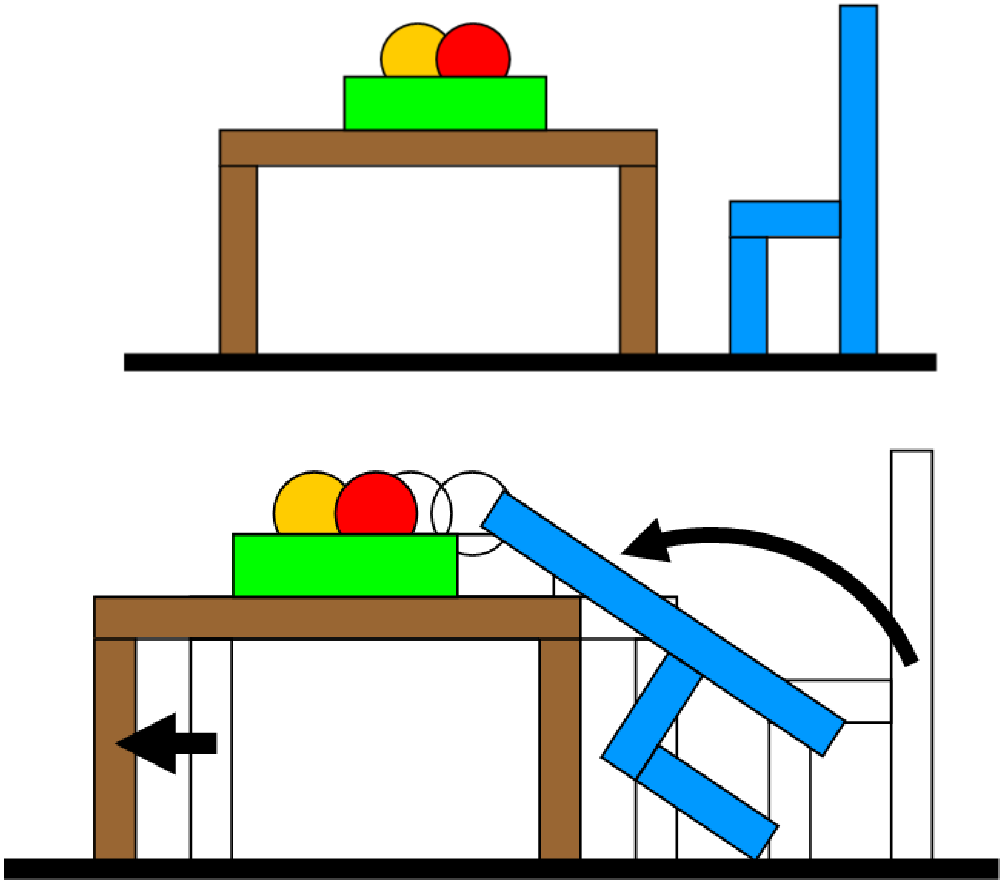


6.1 Hierarchical models

```
Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> } } }
            Material { <BLACK> }
    Plane { <PLANE PARAMS> } }
```

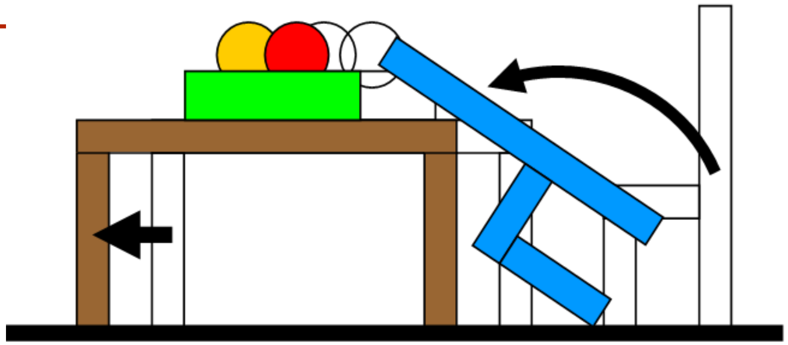


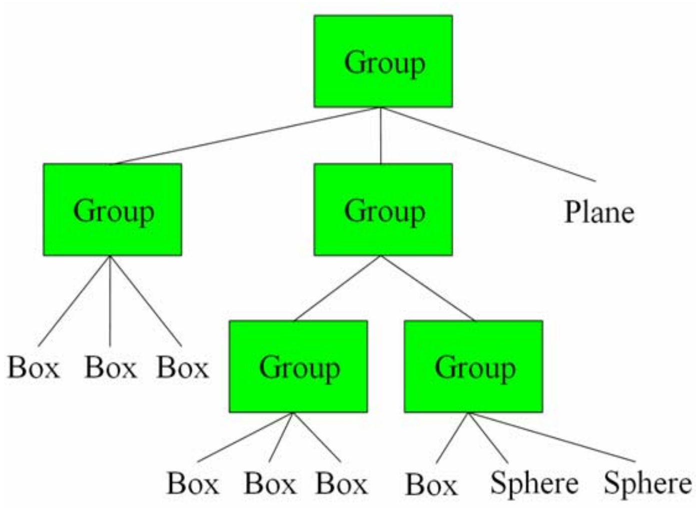
6.1 Hierarchical models

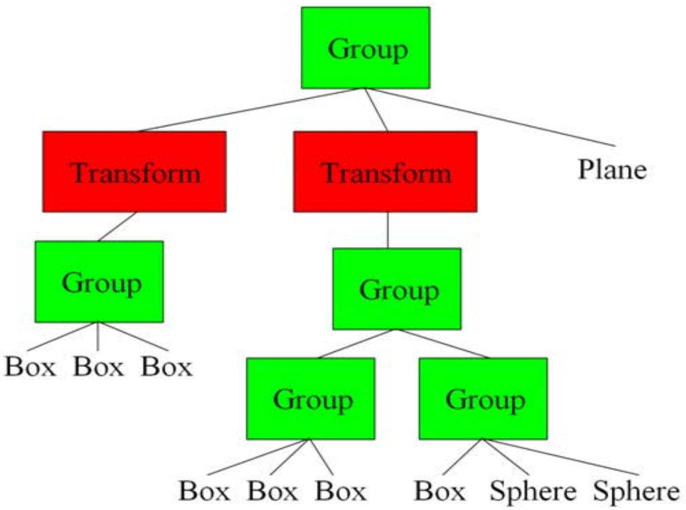


6.1 Hierarchical models

- Transforms position logical groupings of objects within the scene

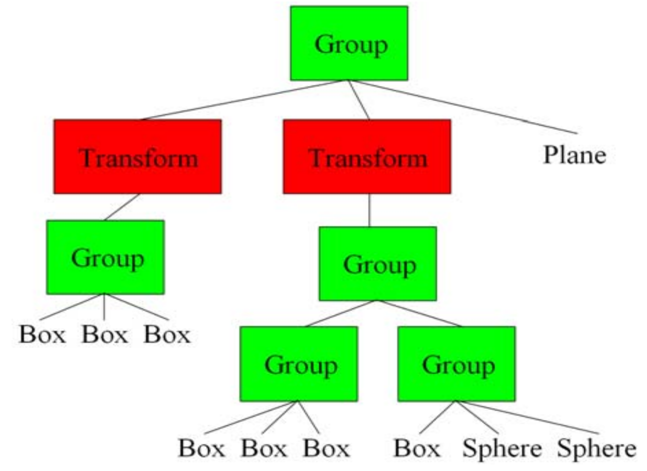






6.1 Hierarchical models

```
Group {
    numObjects 3
    Transform {
        ZRotate { 65 }
        Group {
            numObjects 3
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } } }
    Transform {
        Translate { -2 0 0 }
        Group {
            numObjects 2
            Group {
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> } }
            Group {
                Box { <BOX PARAMS> }
                Sphere { <SPHERE PARAMS> }
                Sphere { <SPHERE PARAMS> } } } }
    Plane { <PLANE PARAMS> } }
```



6.1 Hierarchical models

Note that we have treated translations, rotations, etc. as separate transformations.

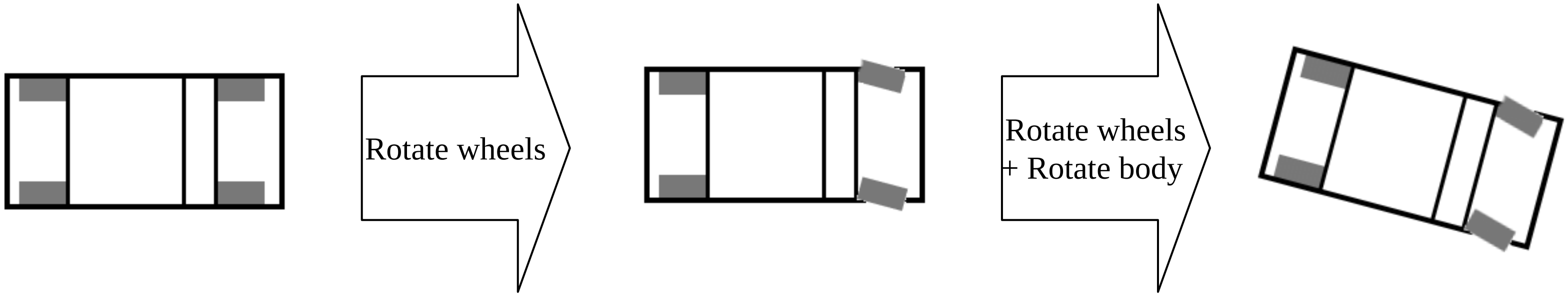
But they are all represented by 3x3 matrices and there is no technical reason not to combine them into the resulting matrix.

It is just simpler for the human programmer, and corresponds to the handle of 3D modeling/animation packages.

It also preserves the relative positions of the different components. Hence, moving the parent object moves all children objects accordingly.

6.1 Hierarchical models

Example



6.1 Hierarchical models

How does OpenGL assist?

Commands to change current transformation:

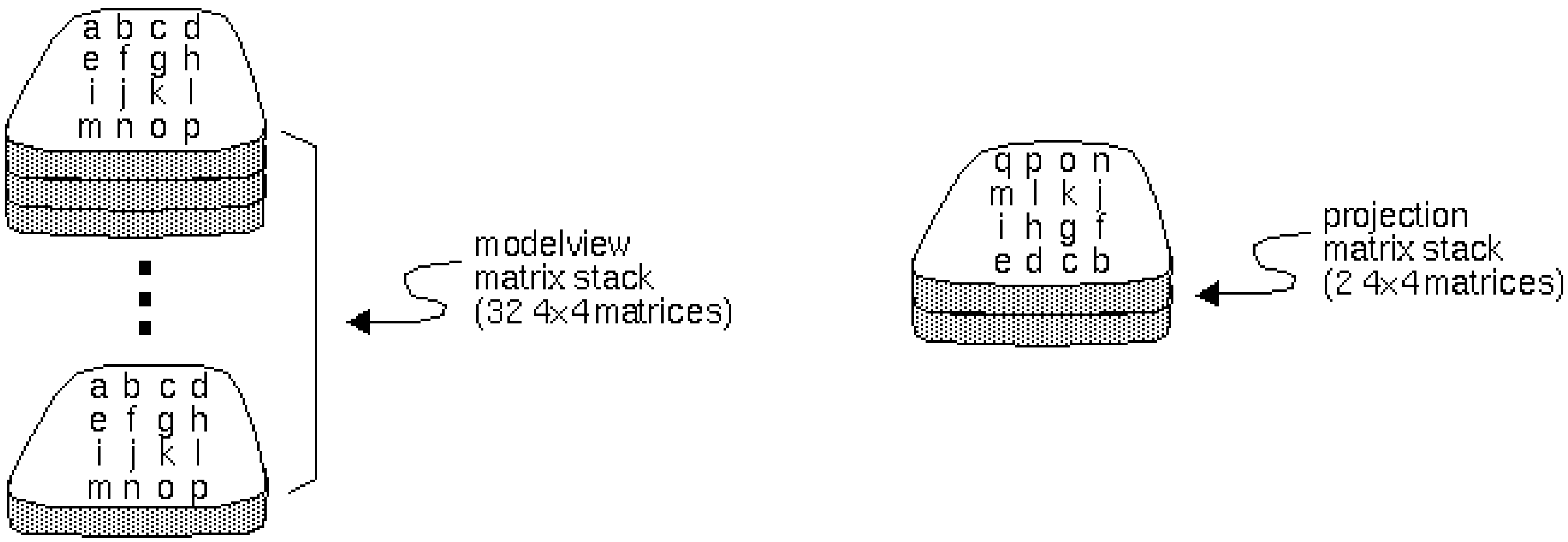
`glTranslate`, `glScale`, etc.

Affects the **state**, i.e. all following commands will undergo this transformation

OpenGL provides methods to maintain a matrix stack, i.e. the capability to revert to previous state.

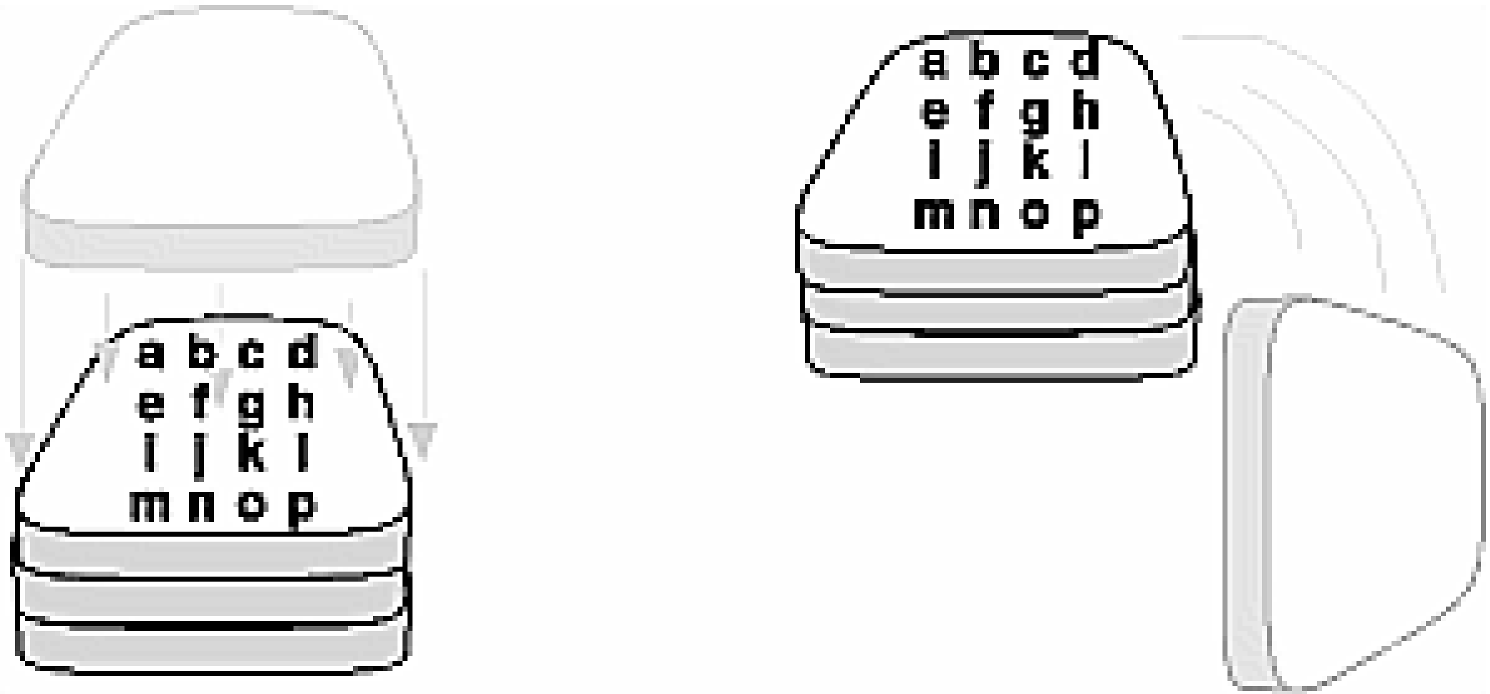
6.1 Hierarchical models

To recall previous transformation, the matrix stack can be used:



6.1 Hierarchical models

`glPushMatrix()   <=>   glPopMatrix()`

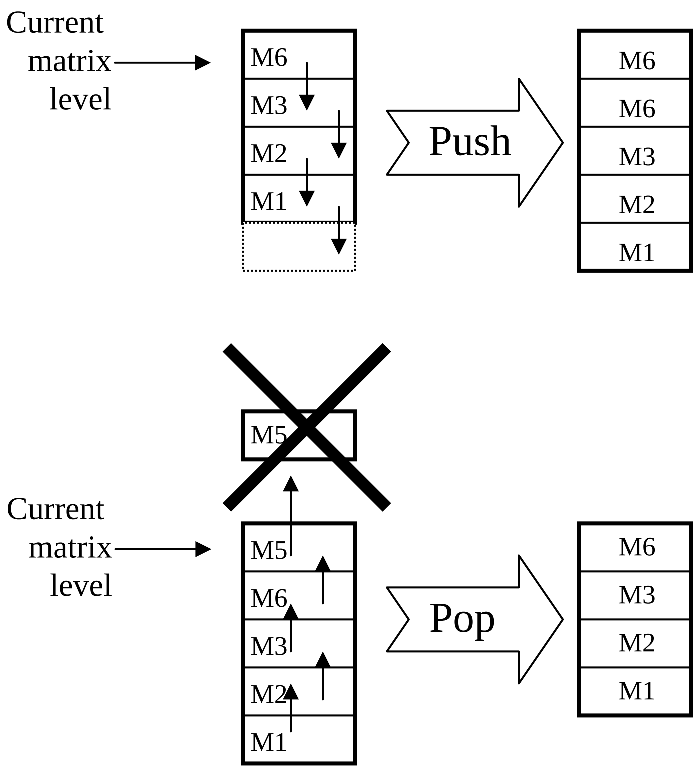


6.1 Hierarchical models

glPushMatrix(void) - Pushes all matrices in the current stack down one level.

glPopMatrix(void) - Pops the top matrix off the current stack, losing the topmost matrix!

(The current stack is determined by **glMatrixMode**).



6.1 Hierarchical models

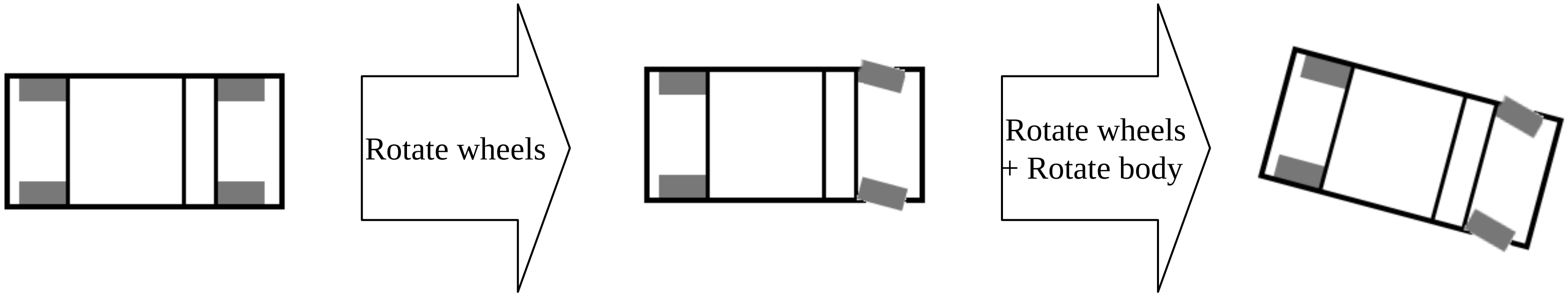
```
draw_body_and_wheel_and_bolts() {
  draw_car_body();
  glPushMatrix();
    glTranslatef(60, 0, 30);
    draw_wheel_and_bolts();
  glPopMatrix();
  glPushMatrix();
    glTranslatef(60, 0, -30);
    draw_wheel_and_bolts();
  glPopMatrix();
  …
}
```

6.1 Hierarchical models

```
draw_wheel_and_bolts() {
  draw_wheel();
  for(i=0;i<5;i++) {
    glPushMatrix();
        glRotatef(72*i, 0, 0, 1);
        glTranslatef(3, 0, 0);
        draw_bolt();
    glPopMatrix();
  }
}
```

6.1 Hierarchical models

Result



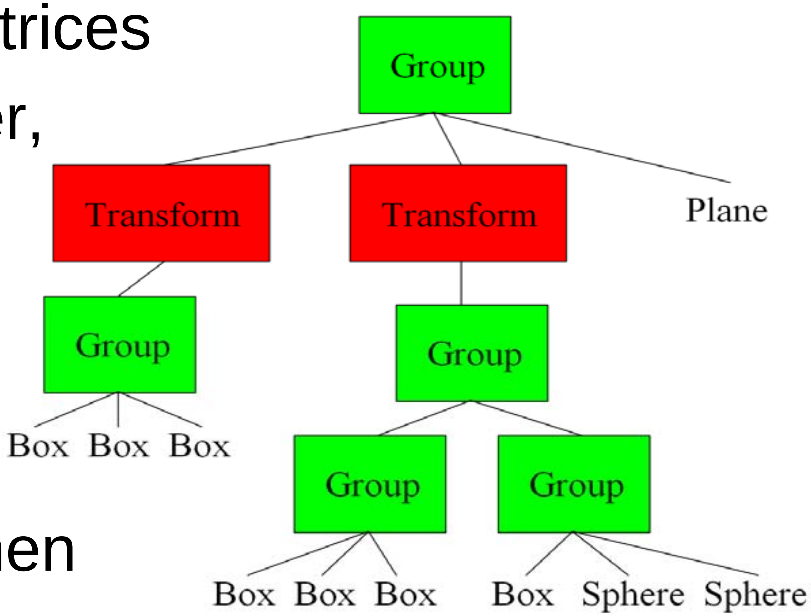
6.1 Hierarchical models

Managing the state

To reset everything: **glLoadIdentity();**

OpenGL stores a stack of matrices

- You don't need to remember, OpenGL remembers
- `glPushMatrix()`
- `glPopMatrix`



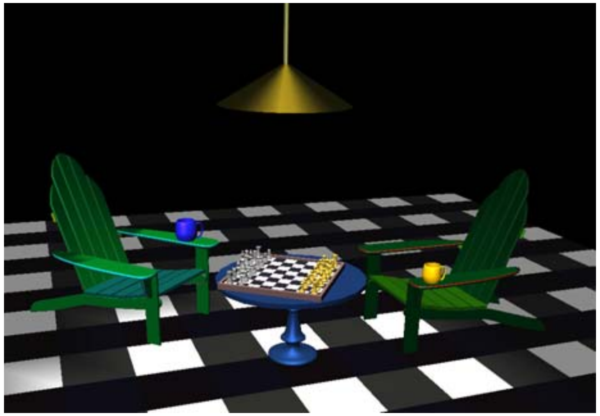
- Typical use: push matrix when you start rendering a group
- Pop once you are done

6.1 Hierarchical models

Display lists

Complex objects can be described in OpenGL using nested display lists to form a hierarchical model. This can be particularly useful if the same object is used several times in the model. For example, the wheels of a car can be stored in a single display list and then drawn four times using the appropriate transformations before calling the display list using **glCallList ()**.

6.1 Hierarchical models



Scene Graph

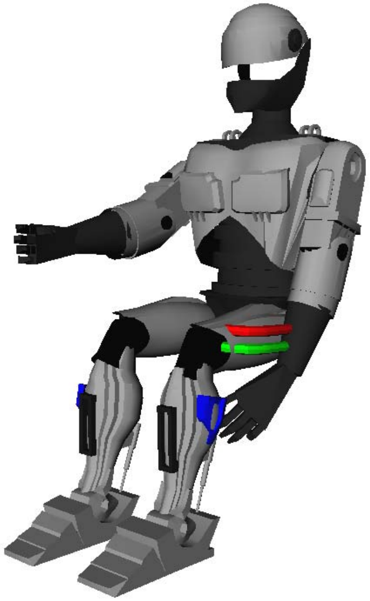
Convenient Data structure
for scene representation:

- Transformations
- Materials, color
- Multiple instances

Basic idea: Hierarchical Tree

Useful for manipulation/animation

Especially for articulated figures



Useful for rendering too

Ray tracing acceleration, occlusion culling

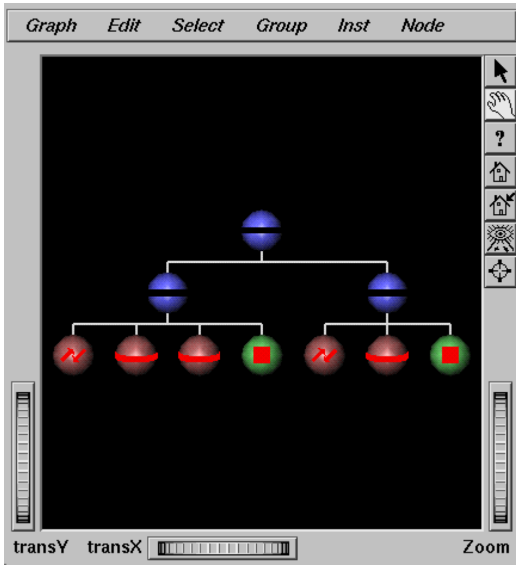
6.1 Hierarchical models

Scene Graph (continued)

Basic idea: Tree

Comprised of several node types:

- Shape: 3D geometric objects
- Transform:
  Affect current transformation
- Property: Appearance, texture, etc.
- Group: Collection of subgraphs



Note: cycles within the tree are not allow (otherwise infinite loops would occur).

6.2 Animation

Hierarchical structure is essential for animation

- Eyes move with head
- Hands move with arms
- Feet move with legs
- …

Without such structure the model falls apart

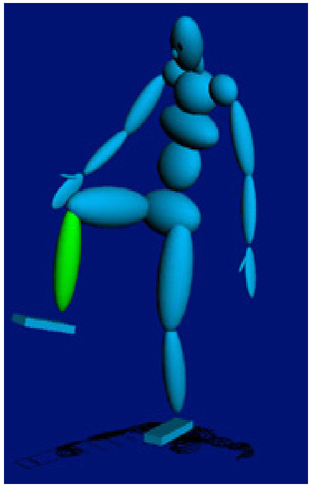
6.2 Animation

Forward kinematics

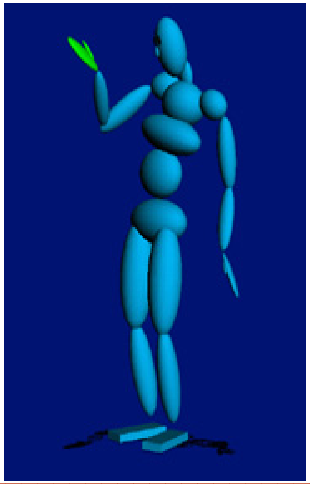
Describes the positions of the body parts as a function of the joint angles.

Each joint is characterized by its degrees of freedom (dof), usually rotation for articulated bodies.

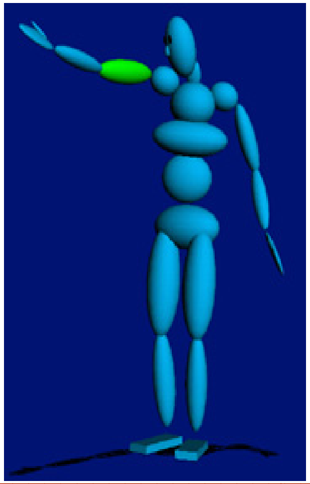
1 DOF: knee



2 DOF: wrist

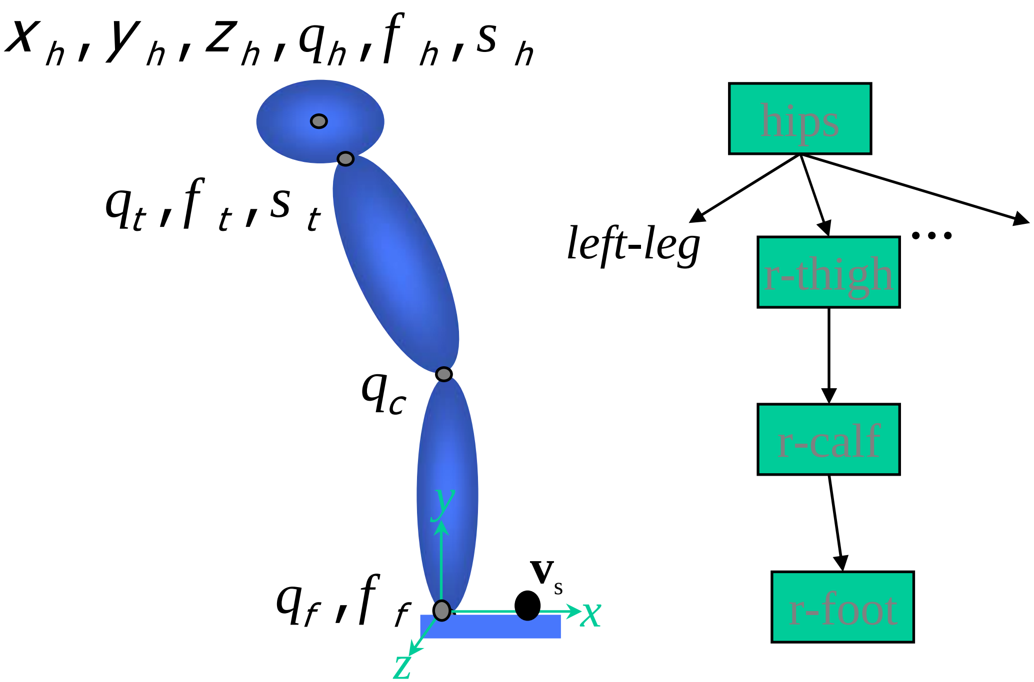


3 DOF: arm



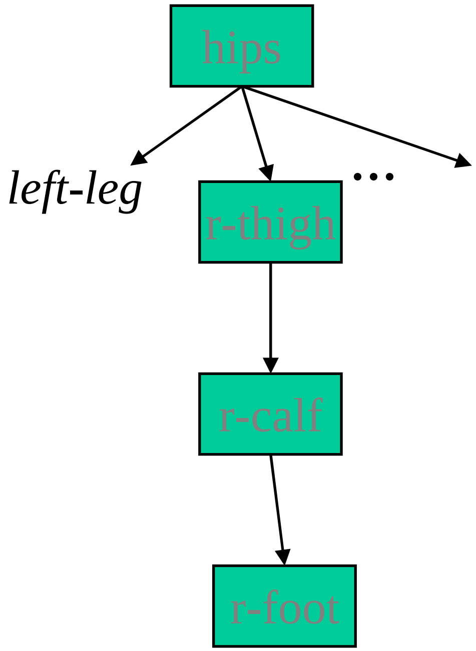
6.2 Animation

Each bone transformation described relative to the parent in the hierarchy:



Derive world coordinates v_w for a point with local coordinates v_s?

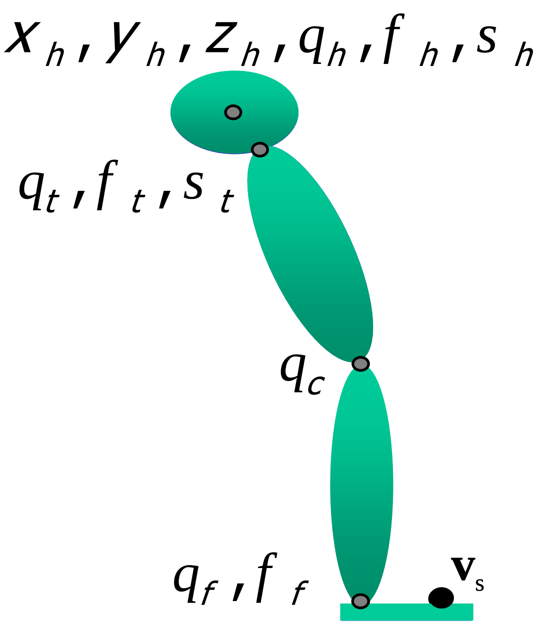
6.2 Animation



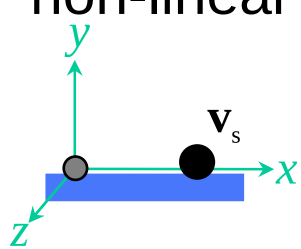
Assumes drawing procedures for thigh, calf, and foot use joint positions as the origin for a drawing coordinate frame.

```
glLoadIdentity();
glPushMatrix();
  glTranslatef(…);
  glRotate(…);
  drawHips();
  glPushMatrix();
    glTranslate(…);
    glRotate(…);
    drawThigh();
    glTranslate(…);
    glRotate(…);
    drawCalf();
    glTranslate(…);
    glRotate(…);
    drawFoot();
  glPopMatrix();
  left-leg
```

6.2 Animation



- Transformation matrix for a point v_s is a matrix composition of all joint transformation between the point and the root of the hierarchy.
- Note that the natural parameters of the degrees of freedom (e.g. angle) have a non-linear effect



$$\mathbf{v}_w = \mathbf{T}(x_h, y_h, z_h)\mathbf{R}(q_h, f_h, s_h)\,\mathbf{R}(q_t, f_t, s_t)\,\mathbf{R}(q_c)\,\mathbf{R}(q_f, f_f)\,\mathbf{v}_s$$

$$\mathrm{v_w} = \mathrm{S}\left(\underbrace{\mathrm{x_h}, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_f, \phi_f}_{p}\right) v_s = \mathrm{S}(p) v_s$$

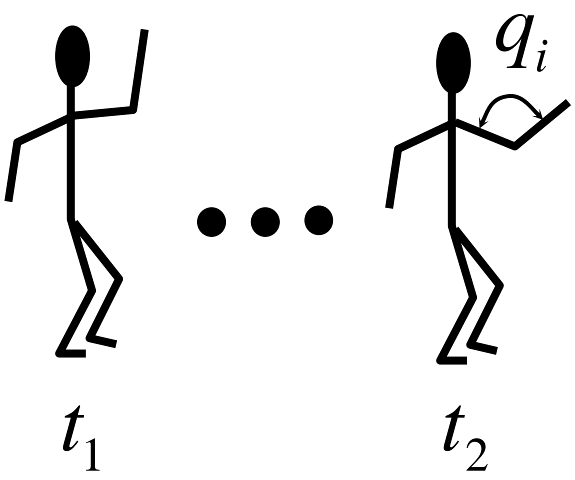
6.2 Animation

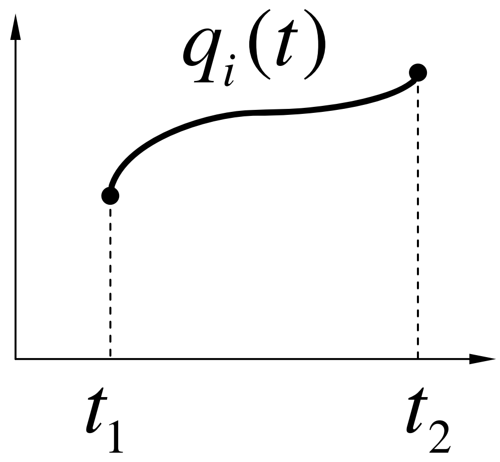
Articulated models

– rigid parts

– connected by joints

They can be animated by specifying the joint angles as functions of time.





6.2 Animation

Example

A stopwatch with second and minute hands.

Hands rotate together as a function of time.

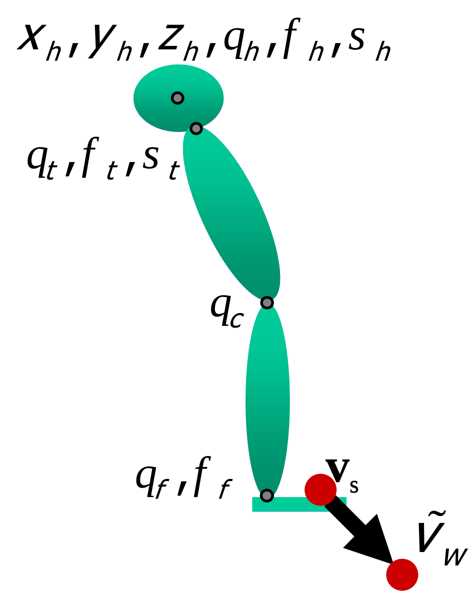
The hands are animated by varying the time parameter.

6.2 Animation

Forward Kinematics

Given the skeleton parameters (position of the root and the joint angles) *p* and the position of the point in local coordinates v_s, what is the position of the point in the world coordinates v_w?

Not too hard, we can solve it by evaluating $S(p)v_s$



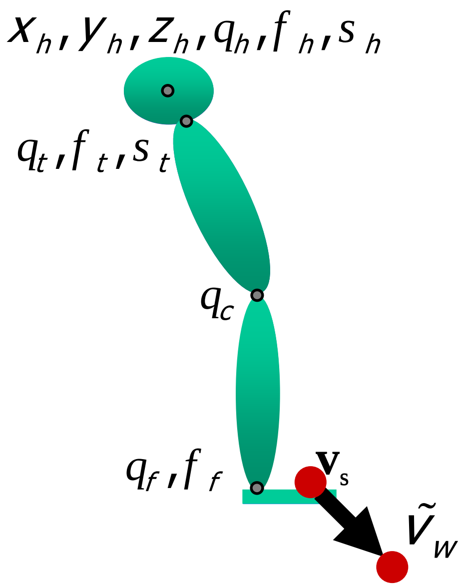
6.2 Animation

Inverse Kinematics

Given the position of the point in local coordinates v_s and the desired position $\tilde{v}_w$ in world coordinates, what are the skeleton parameters *p?*

This problem is much harder to solve since it requires solving the inverse of the non-linear function:

In addition, it is an underdetermined problem with many solutions.



6.2 Animation

Example

Simple geometric example (in 3D):
in order to specify hand position, need elbow and shoulder:

The set of possible elbow location is a circle in 3D.