COMP 3069
Computer Graphics

Lecture 4:
Hierarchical Modelling

Autumn 2018

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Today's Content

- OpenGL transformation matrix stacks
  - Store transformation matrices
- Push/pop transformation matrix stack
  - Save current scene state for come back later
- Hierarchical modelling
  - Creating complex object/scene
- Pixar lamp modelling, animation, and demo

OpenGL Matrix Stacks

- There are 3 **transformation matrix stacks:**
  - Model-View - glMatrixMode(GL_MODELVIEW);
  - Projection - glMatrixMode(GL_PROJECTION);
  - Texture - glMatrixMode(GL_TEXTURE);
- In Model-View mode, an identity matrix is created, which becomes the ‘current matrix’ and sits on top of the matrix stack
- When another transformation is called in the program, the current matrix is updated, e.g., multiplied (from the right) by the new matrix corresponding to the transformation called.

OpenGL Matrix Stacks

- Say you call glMatrixMode(GL_MODELVIEW)
- If you then call glTranslatef(0,0,-250), the current matrix will be

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- If you call glTranslatef(0,0,-250) again, current matrix will be:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -500 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- If you then call DrawCube() – the cube will be drawn at a distance of 500 from the origin, in the -Z direction

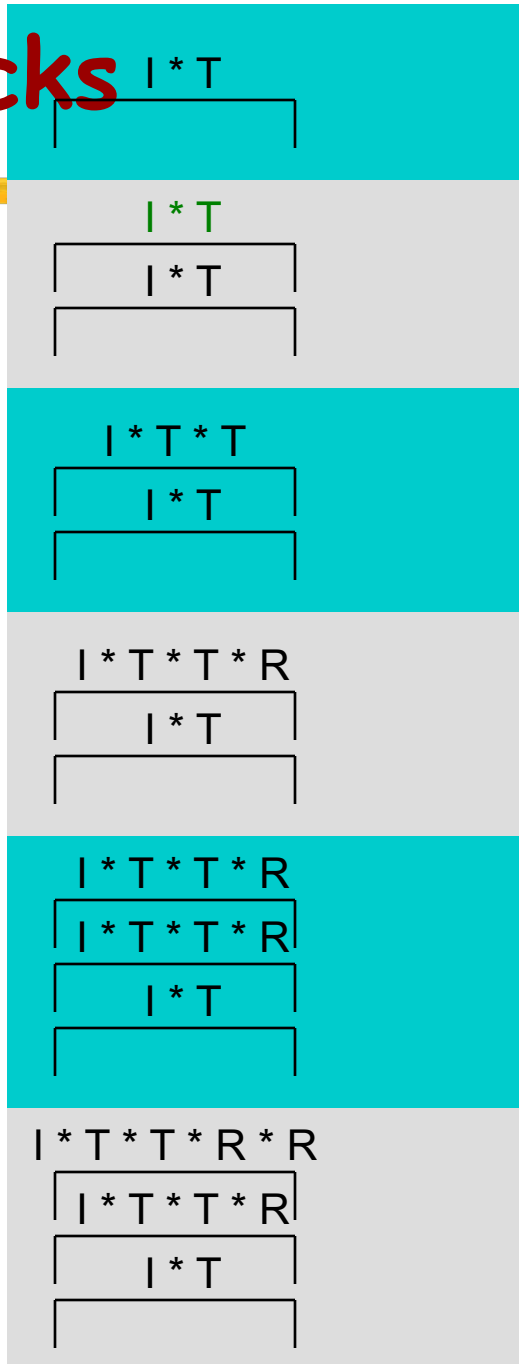
Push/Pop Matrix Stacks

- If at some point you need to do something somewhere in your object / scene, but you know you will be back to the current position later on, you should call **glPushMatrix()**, which will make a copy of the current matrix and push it on top of the matrix stack.

- When you are ready to go back to where you were, you should use **glPopMatrix(),** which will 'pop' the saved matrix off the stack, and you are back where you were.

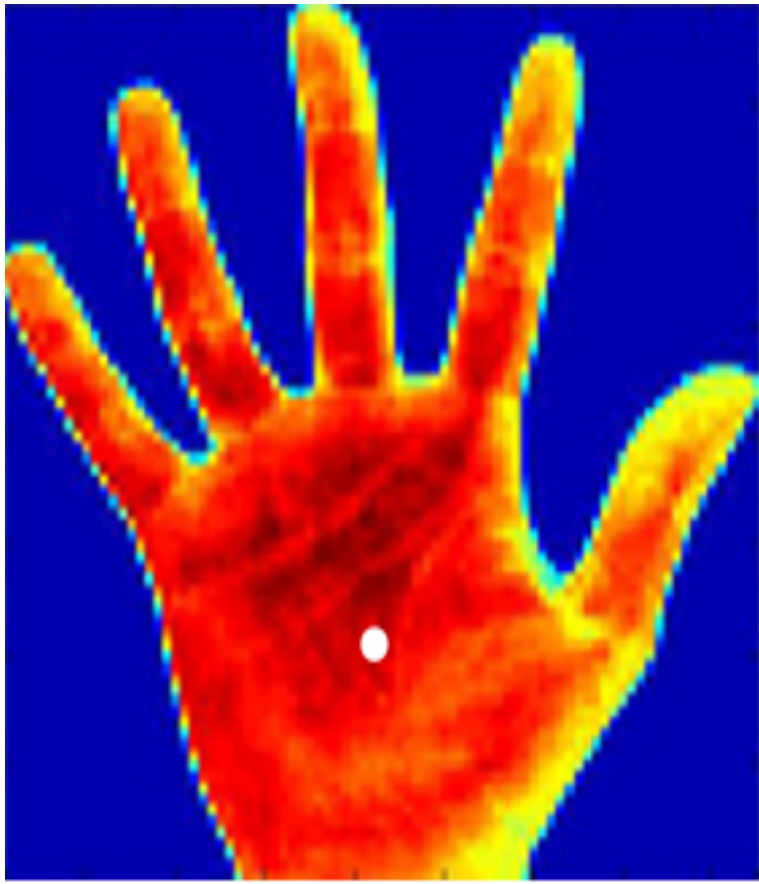
Push/Pop Matrix Stacks

- The modelling stack in action:
  - glTranslatef(10,0,0);
  - glPushMatrix();
    - glTranslatef(10,0,0);
    - glRotatef(45,0,1,0);
  - glPushMatrix();
    - glRotatef(45,0,1,0);
  - glPopMatrix();
  - glPopMatrix();



Using glPushMatrix / glPopMatrix

- Say you are at the centre of the palm (CoP)
- **glPushMatrix()** saves a copy of current matrix
- Rotate and translate to draw thumb as a cylinder
- **glPopMatrix()** so you are at the CoP again
- **glPushMatrix()** to save a copy of current matrix
- Rotate and translate to draw index finger
- **glPopMatrix()** so you are at the CoP again, to draw middle finger ..

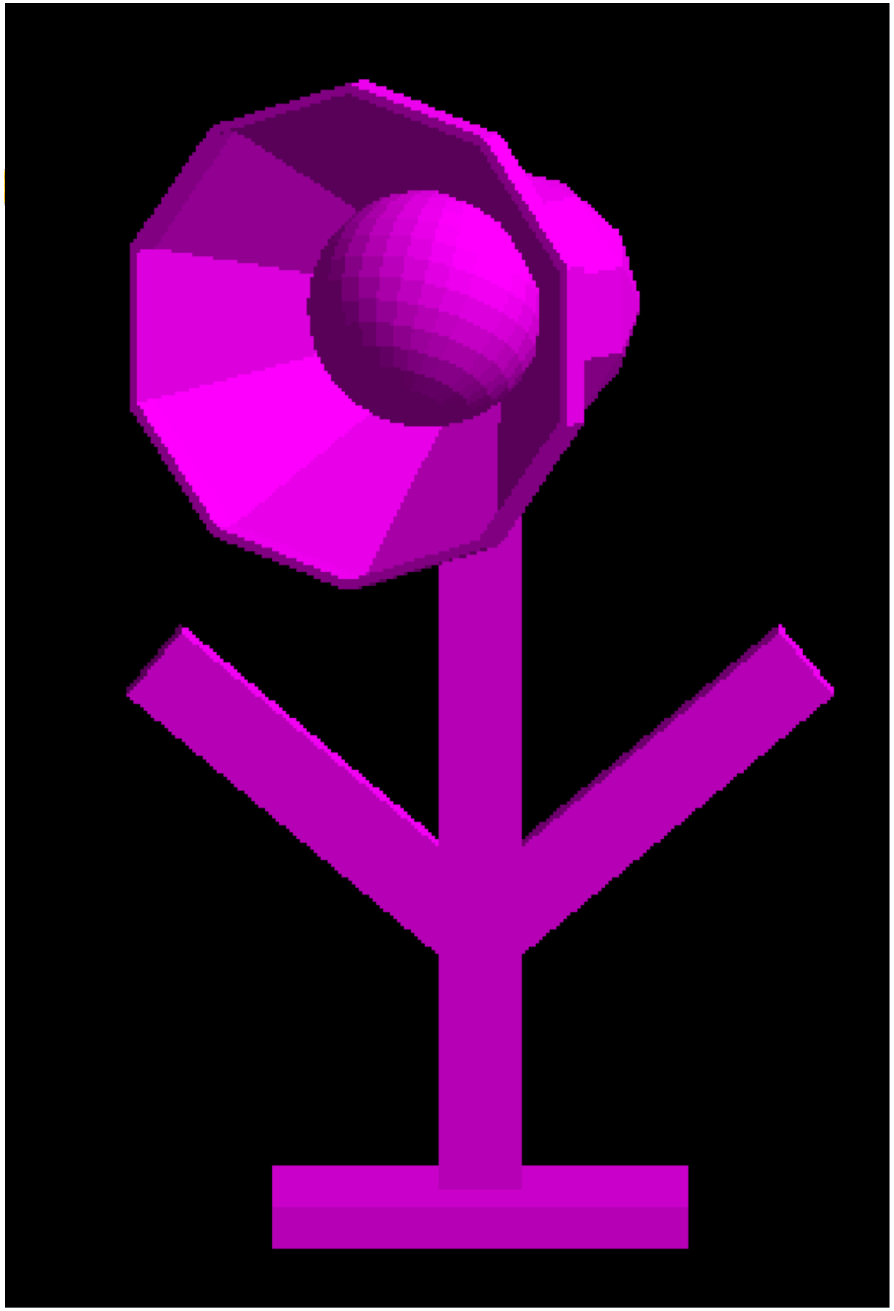


Using glPushMatrix / glPopMatrix

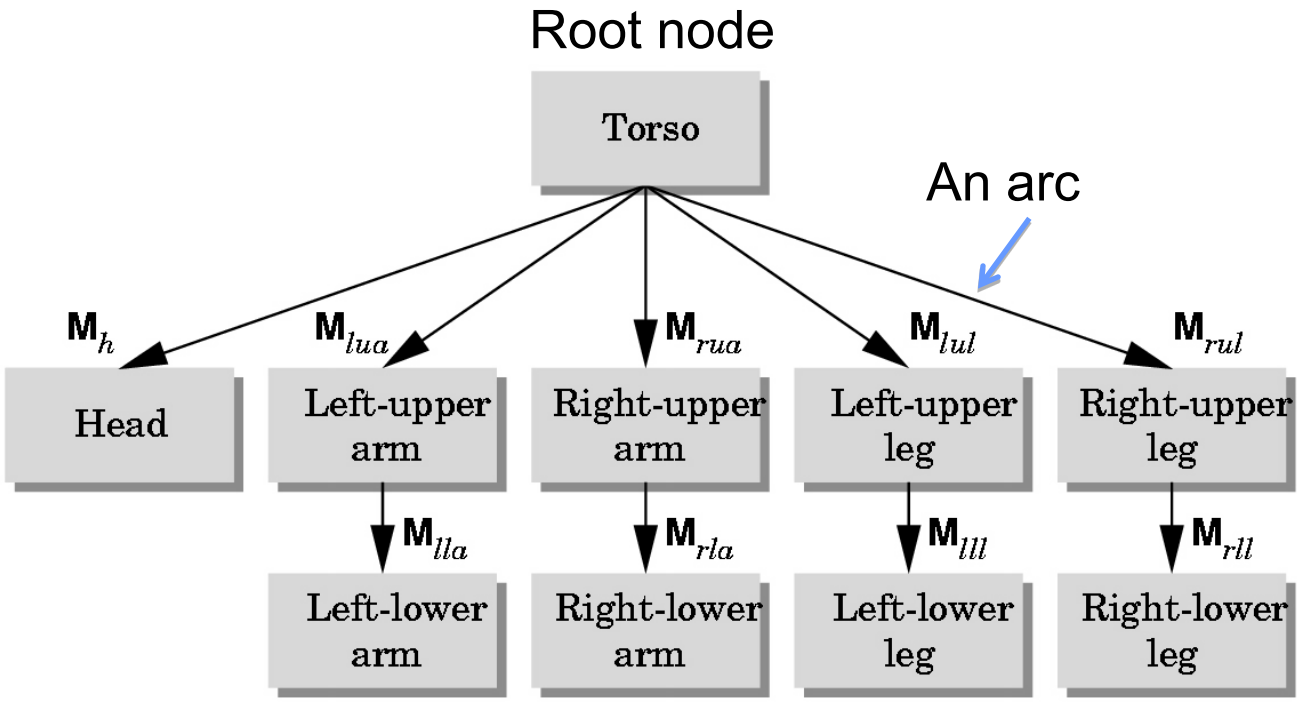
The lamp has a base, a10 inch lower arm and a 10 inch upper arm, and a head.

```
DrawBase();
glTranslated(0.0, 5.0, 0.0);
DrawArm();
glTranslated(0.0, 10.0, 0.0);
DrawArm();
glTranslated(0.0, 5.0, 0.0);
glRotated(-25.0, 0.0, 1.0, 0.0);
DrawHead();
```

```
DrawBase();
glTranslated(0.0, 5.0, 0.0);
DrawArm();
glTranslated(0.0, 5.0, 0.0);
glPushMatrix();
glTranslated(5, 0.0, 0.0);
glRotated(-50.0, 0, 0, 1);
DrawArm();
glPopMatrix();
glPushMatrix();
glTranslated(-5, 0.0, 0.0);
glRotated(50.0, 0, 0, 1);
DrawArm();
glPopMatrix();
glTranslated(0.0, 5.0, 0.0);
DrawArm();
glTranslated(0.0, 5.0, 0.0);
glRotated(-25.0, 0.0, 1.0, 0.0);
DrawHead();
```

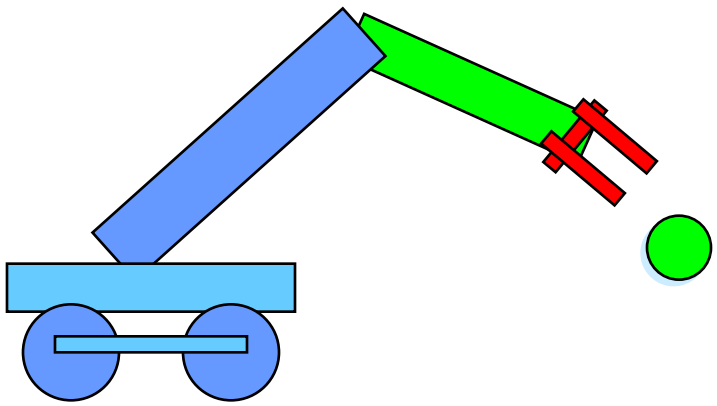


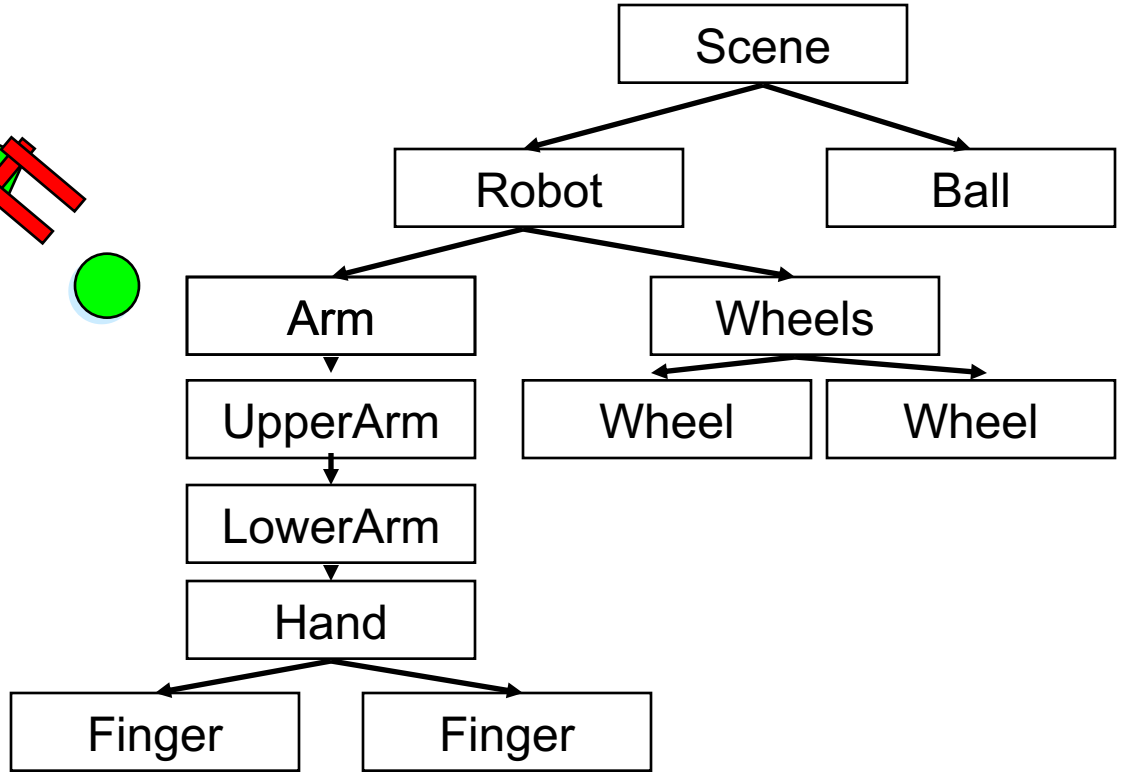
Hierarchical Modelling



- Human body is hierarchical, with body, head, upper arm, lower arm, etc…

Hierarchical Modeling

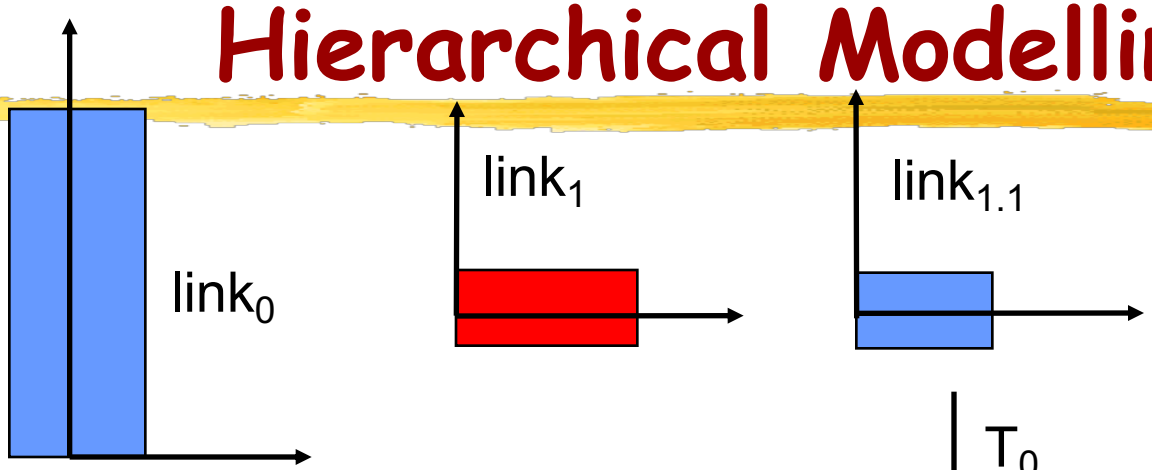




Data Structure for Hierarchical Modelling

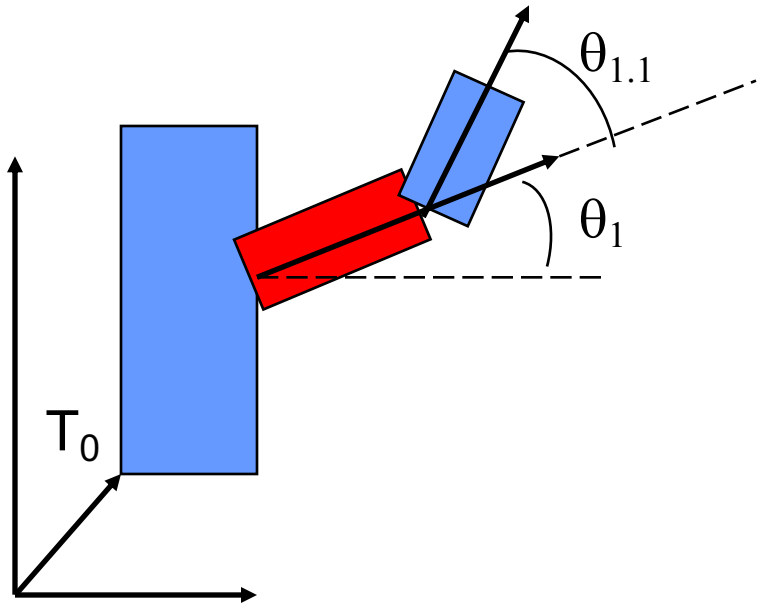
- Generally represented as a tree, with nodes and arcs
- A **node** is a link or a part of the object
- An **arc** contains transformation to be applied to the node, and the rest of the linkage further down the hierarchy
- It is convenient to define points in the **local coordinate system** associated with a joint and to have a method for converting the coordinates of a point from one system to another

Data Structure for Hierarchical Modelling



T_0

Data for $link_0$



T_1 transforms link1 relative to link0

$R_1(\theta_1)$

Data for $link_1$

$T_{1.1}$ transform link1.1 relative to link1

$R_{1.1}(\theta_{1.1})$

Data for $link_{1.1}$

Kinematics

- **Kinematics** is the study of object movements irrespective of their speed or style of movement
- **Forward kinematics** traverses the tree structure and propagates transformations downward - vertex of an object can be transformed by concatenating all transformations higher up in the tree and applying the composite transformation to the vertex. High DOF models are tedious to control this way
- **Inverse kinematics** provides a "goal-directed" approach to animating a character, e.g., given the position of an object, how to configure the robot arm to reach that object

Keyframe Animation Techniques

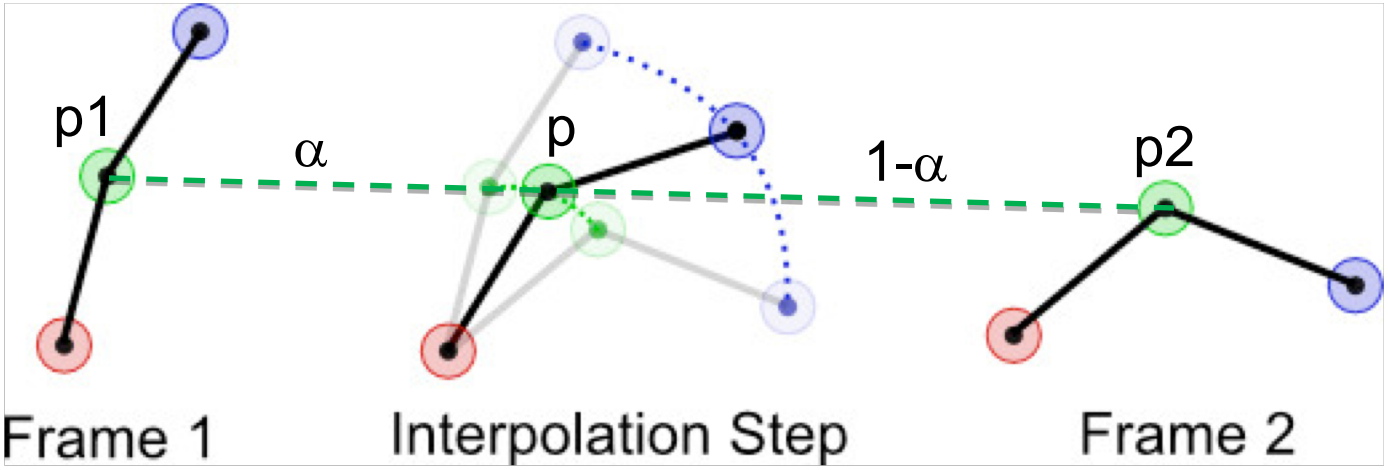
- Commonly used and easier than kinematics.
- Artist draws the major frames of the animation.
- Major frames are the ones in which prominent changes take place. Animator specifies critical or key positions for the objects, and speed of animation etc.
- The computer then automatically fills in the missing frames by smoothly interpolating between those positions.

Interpolation-based: Keyframe Animation

- The keyframe of a model contains information such as the model's structure, position, orientation, and scale, e.g., 3 key frames:

Interpolation based: Keyframe Animation

- Suppose we have two keyframes that contain a bone structure with three joints (red, green and blue vertices)
- We render frame one, then an intermediate scene (obtained through interpolation) before rendering frame two for a more smooth animation flow



e.g., linear interpolation: **p = (1-α) p1 + α p2**

```
void Lamp::Update( const double& deltaTime )
{
if( animateTime < 1.0 ) {
        if( keyframe != 0 ) {
            keyframe = 0;
            animateTime = 0.0;
            animateRotation = 0.0;
            animateTranslation = 0.0;
            interpA = -45.0;
            interpB = -60.0;
            interpTime = 1.0;
        }
    }//have now interpolated to keyframe 1
    else if( animateTime < 1.25 )
    {
        if( keyframe != 1 ) {
            keyframe = 1;
            animateRotation = 0.0;
            interpA = -60.0;
            interpB = -15.0;
            interpTime = 0.25;
        }
animateTranslation = -(4.0*9.81)*(animateTime-1.0) + 0.5*(animateTime-1.0)*(animateTime-1.0)*(4.0*GRAVITY);
    } // have now interpolated to keyframe 2

else {
        if( keyframe != 2 ) {
            keyframe = 2;
            animateRotation = 0.0;
            interpA = -15.0;
            interpB = -45.0;
            interpTime = 1.75;
        }

animateTranslation = -(4.0*9.81)*(animateTime-1.0) + 0.5*(animateTime-1.0)*(animateTime-1.0)*(4.0*GRAVITY);
    }
}
```

- gluLookAt(0.0, 5.0, 0.0, 0.0, 0.0, -50.0, 0.0, 1.0, 0.0);
- glTranslated(0.0, 0.0, -500.0);
- glScalef(12.0, 12.0, 12.0);
- glTranslated(0, -25.0+animateTranslation, 0.0); // for jumping up and down
- DrawBase();

// the 1st (negative) rotation point is at the centre of the base, the start of the lower arm

- glRotated(interpA + animateRotation*((interpB-interpA)/interpTime), 0.0, 0.0, 1.0);
- glTranslated(0.0, 7.0, 0.0); // move to the centre of the lower arm
- DrawArm(); // draw the lower arm

// move to the 2nd (positive) rotation point, from the centre to the end of the lower arm

- glTranslated(0.0, 7, 0.0);
- glRotated(2.0*(- (interpA + animateRotation*((interpB-interpA)/interpTime))), 0.0, 0.0, 1.0);
- glTranslated(0.0, 7.0, 0.0); // move to the centre of the upper arm
- DrawArm();
- glRotated(-25.0, 0.0, 1.0, 0.0);
- glTranslated(0.0, 5.0, 0.0);
- DrawHead();