CSCI 4611

Character animation using mocap data

Midterm

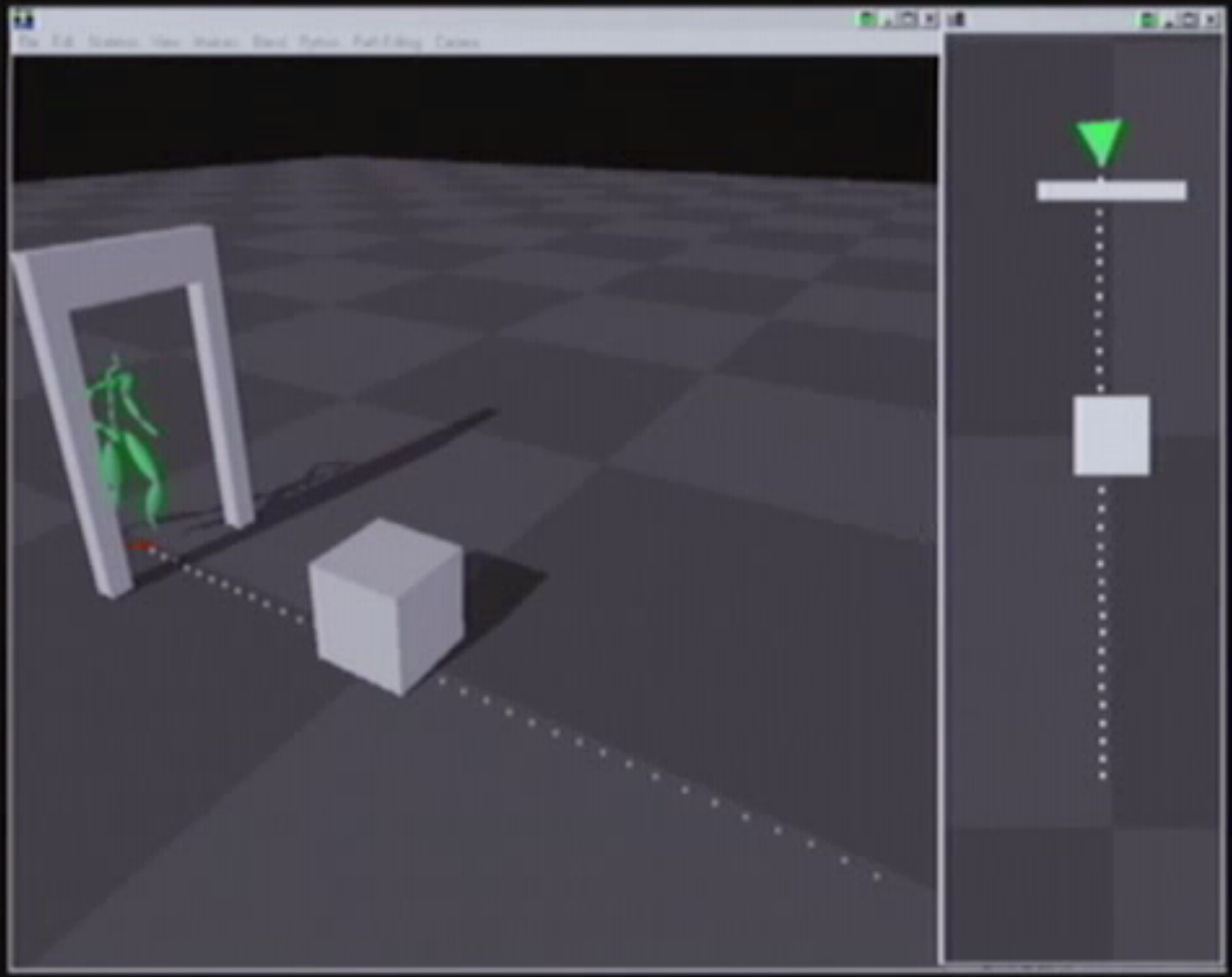
Grades will be posted on Wednesday

We’ll discuss the questions and answers in class.

Assignments

- Assignment 3 due tonight

- Assignment 4 will be posted soon (may be delayed by a day or two)



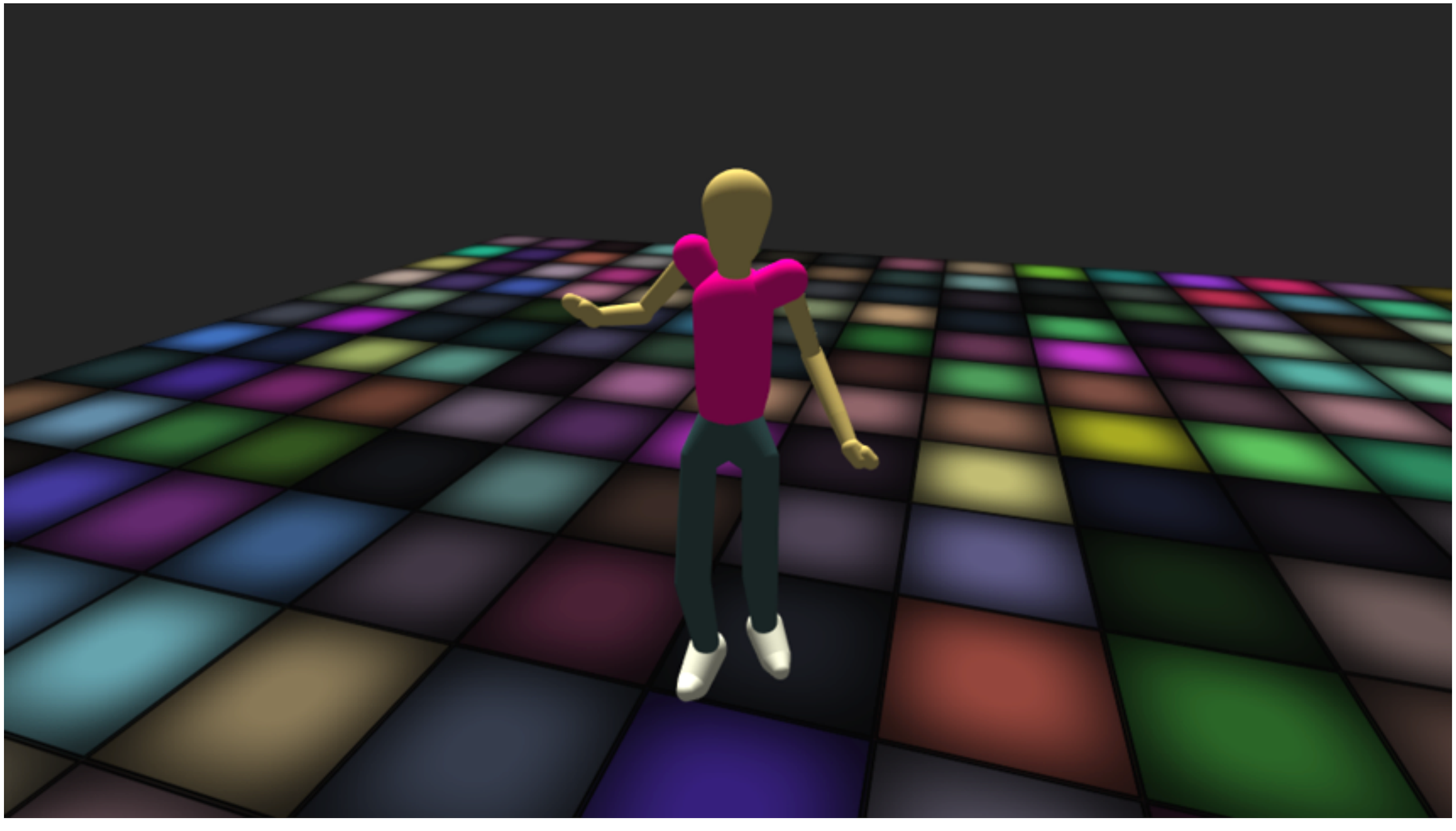
Mocap data

In Assignment 4, we'll be working with data from the CMU Motion Capture Database (http://mocap.cs.cmu.edu/)

- 2,605 different motions

Here are a few examples…

Last year: The Macarena



Assignment details

We will give you code to load and parse mocap data files.

You will need to:

- work with the hierarchical model to render the character properly.

- adjust the transformations and animation speed to make the character walk along a specified curve.

CMU mocap format

Two kinds of files:

- `.asf` files specify skeleton
- `.amc` files specify motions

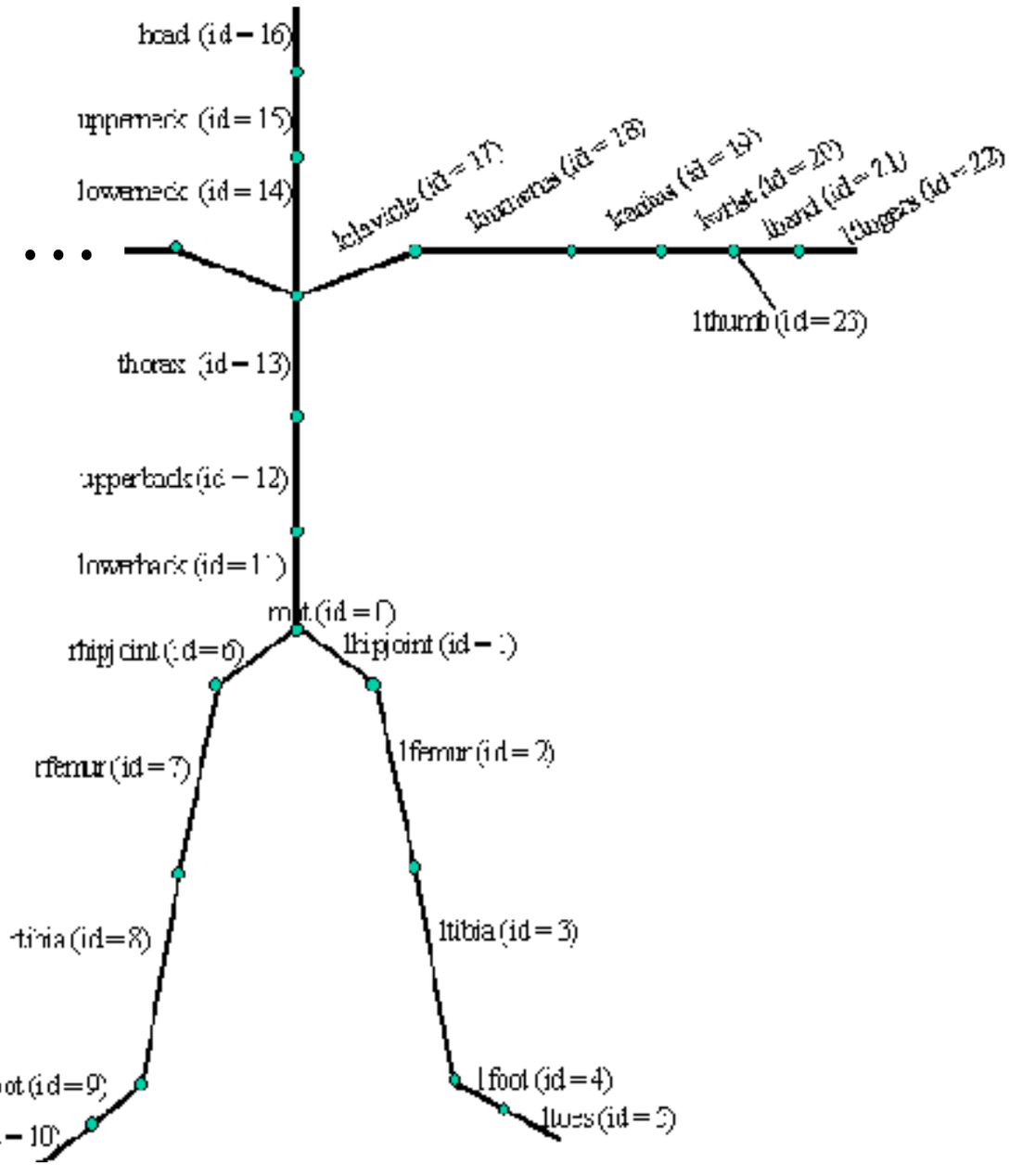
Multiple `.amc` files can use the same `.asf` file.

CMU mocap skeleton

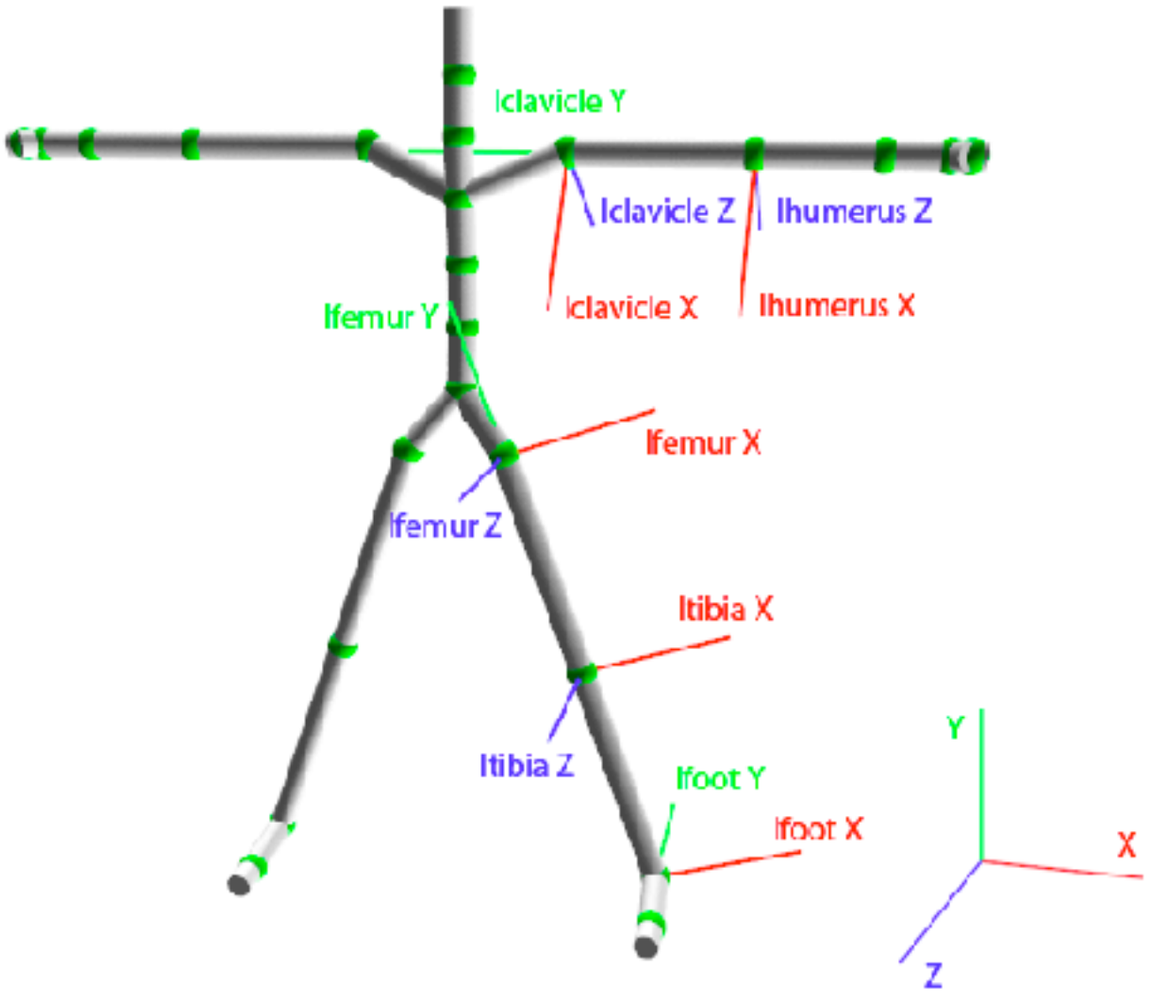
Each “bone” has

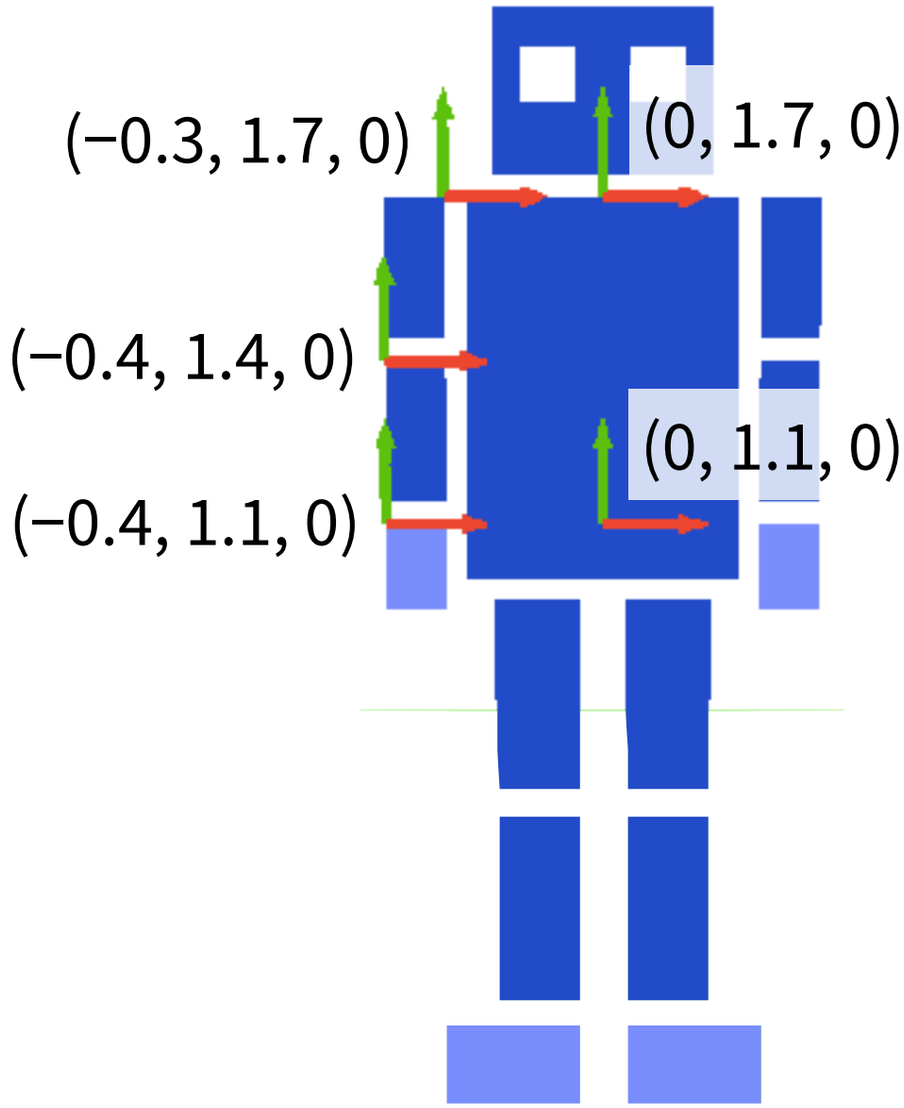
- a rest coordinate frame
- a direction (in its local coordinate frame)
- a length
- zero or more children

Root node is the origin of the skeleton



Local coordinate frames





The `Character` class

This is like the root node of the skeleton.

- Already contains code to load the entire skeleton and the animation data.

- The skeleton is a hierarchy: The character stores a list of `Bones` attached to the root node. Each `Bone` in turn has a list of other `Bones` attached to it.

- The character also stores the translation and rotation of the root node.

You need to draw all the bones attached to the root node. Each one should recursively draw all of its child bones.

The `Bone` class

Represents a single bone in the skeleton.

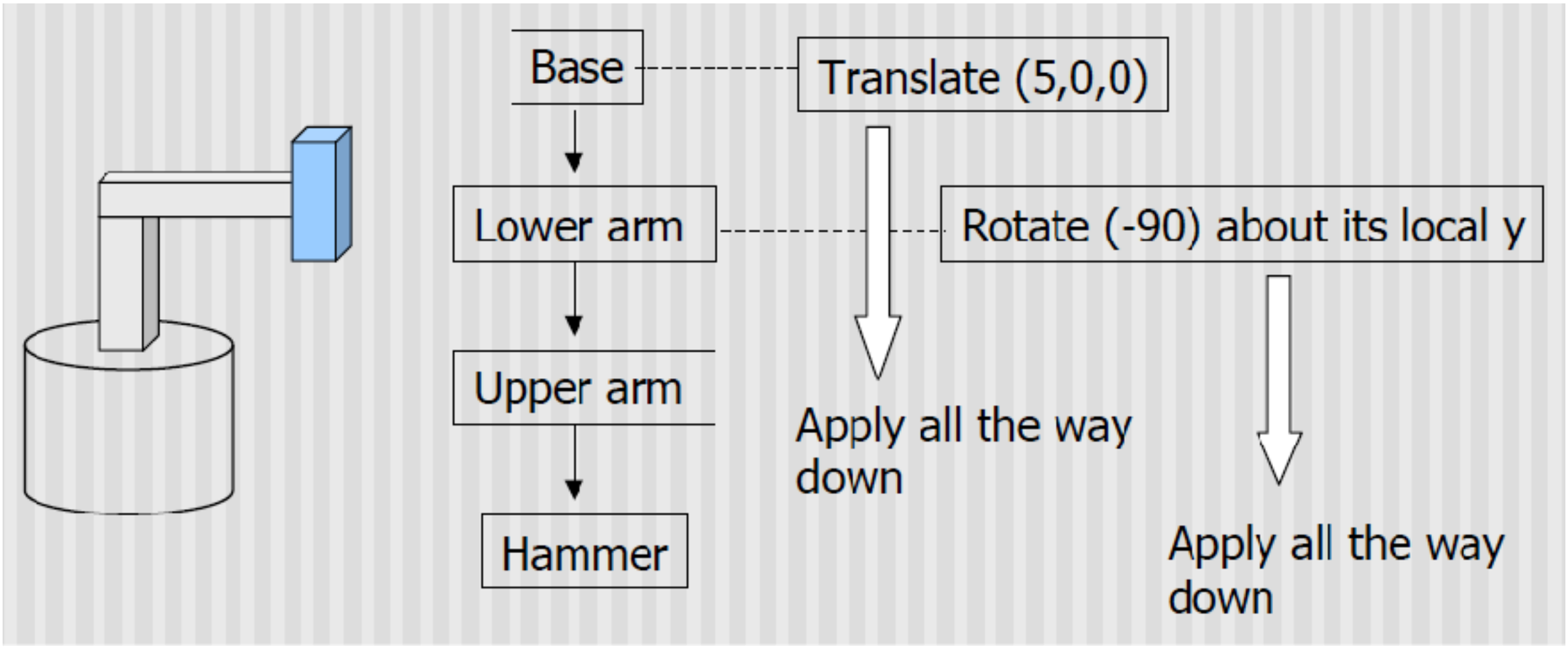
When the skeleton is being animated, this class will automatically update with new bone rotation data.

- There is a method `getCurrentLocalRotation()` that returns current rotation as a `glm::mat4` (a 4×4 matrix).

- The length and resting orientation of the bone are found using the `getBoneVector()` method. Think of the bone as beginning at the origin in its local coordinate frame, and ending at `getBoneVector()`.

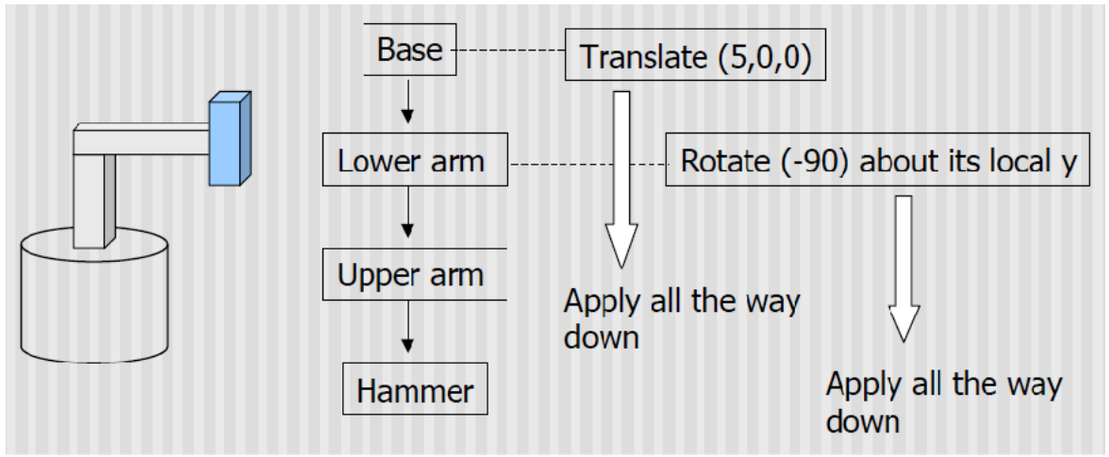
- After applying the current rotation for the bone, you can draw a line segment or cylinder between these two points to draw the bone.

Hierarchical transformations

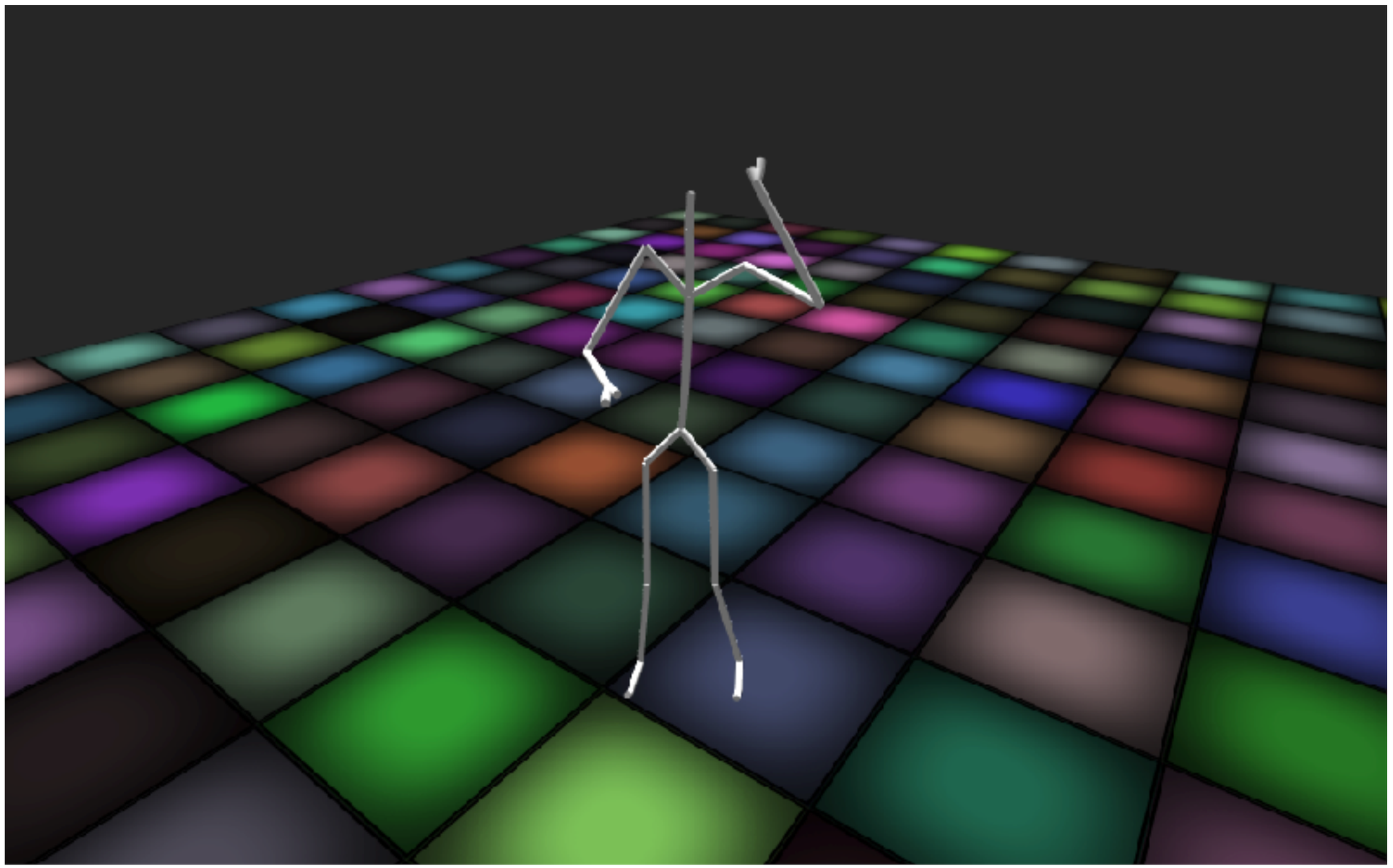


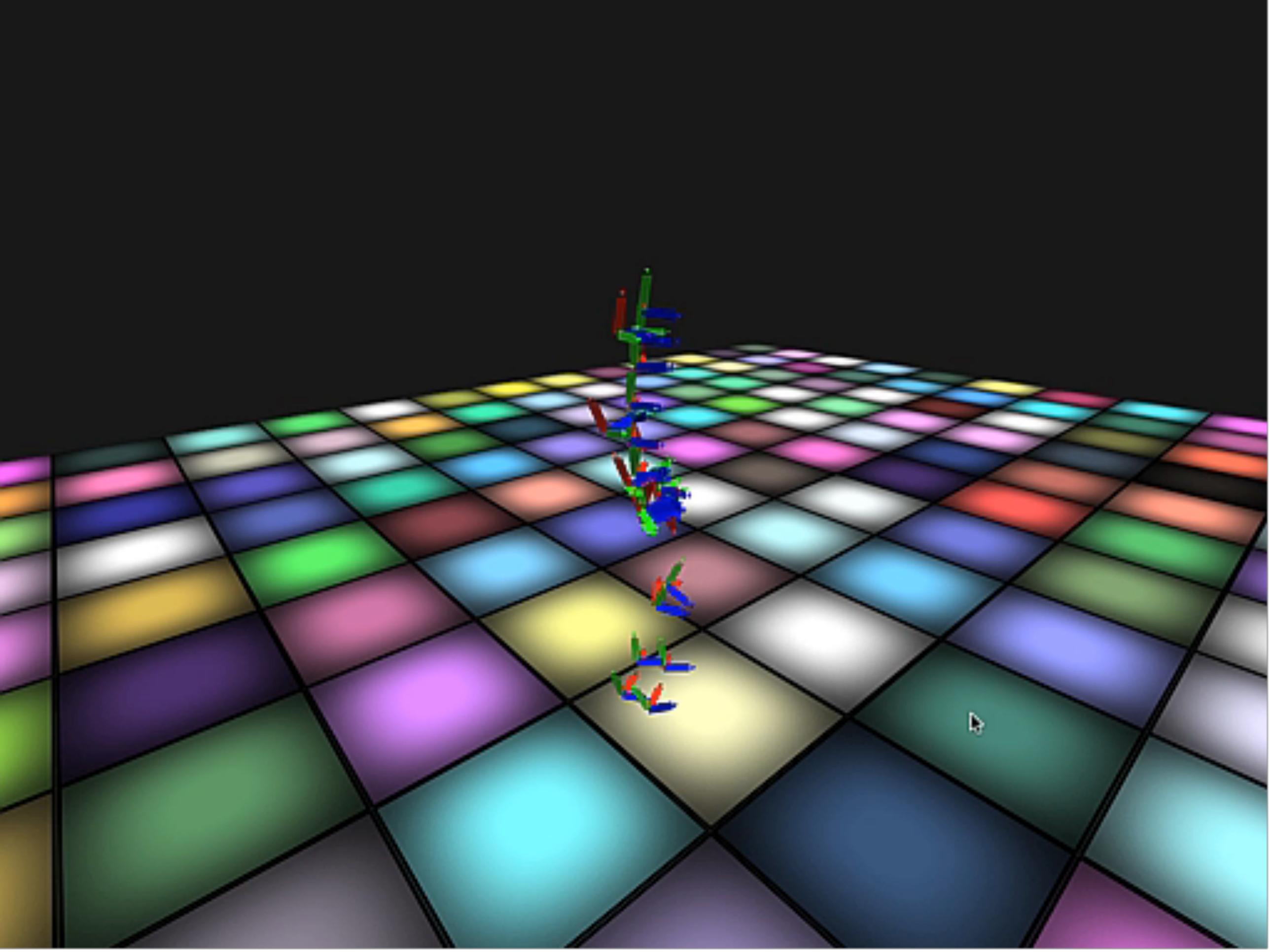
The matrix stack

- Push matrix
  - Apply base translation
  - Draw base
  - Push matrix
    - Apply lower arm rotation
    - Draw lower arm
    - [Draw descendants]
  - Pop matrix
- Pop matrix



Bones as cylinders





Useful GLM functions

```
using glm::vec3; // otherwise type glm::vec3

vec3 sum = v1 + v2; // can add, subtract
vec3 twice = 2.0f*v1; // must multiply by float
```

```
float d = glm::dot(v1, v2);
vec3 c = glm::cross(v1, v2);
float len = glm::length(v); // norm
vec3 vhat = glm::normalize(v); // unit vector

float deg = glm::degrees(rad);
float rad = glm::radians(deg);
```

Useful GLM functions

```
using glm::mat4;

mat4 m1; // initialized to identity
mat4 m = m1*m2 + m3; // can add, subtract, multiply
```

```
float det = glm::determinant(m);
mat4 minv = glm::inverse(m);
mat4 mtrans = glm::transpose(m);

#include <glm/gtc/matrix_transform.hpp>

mat4 mt = glm::translate(m, v);
mat4 mr = glm::rotate(m, angle, v); // angle in deg
mat4 ms = glm::scale(m, v);
// all three return m times transformation matrix
```

GLM → OpenGL

So you've computed a transformation in a `mat4`.
Apply it to the current OpenGL transformation:
`glMultMatrixf(&m[0][0]);`

So for example, these two things are equivalent:

1.
```
glTranslatef(1, 2, 3);
```

2.
```
mat4 t = glm::translate(mat4(), vec3(1,2,3));
glMultMatrixf(&t[0][0]);
```

Next class

Curves and surfaces