Hierarchical Transformation
Summary

• If we think:
  • Each transformation updates the vertex in an absolute world.
  • Then, the code is arranged in a reverse order.

• Alternatively, OpenGL thinks:
  • A transformation updates the coordinate system.
  • For each change, the transformation is defined in the current (local) coordinate system.
  • Transformation matrix and coordinate system are equivalent.
  • The code is in the forward order!
A hierarchy

```c
glLoadIdentity();
// A dot at 0,0,0
```
A hierarchy

glLoadIdentity();
// A dot at 0,0,0
glTranslatef(2,0,0);
// A dot at 0,0,0
A hierarchy

```c
glLoadIdentity();
// A dot at 0,0,0
glTranslatef(2,0,0);
// A dot at 0,0,0
glTranslatef(1,1,0);
glRotatef(45,0,0,1);
// A dot at 0,0,0
```
A hierarchy

glLoadIdentity();
// A dot at 0,0,0
glTranslatef(2,0,0);
// A dot at 0,0,0
glTranslatef(1,1,0);
glRotatef(45,0,0,1);
// A dot at 0,0,0

A hierarchy

glLoadIdentity(); // A dot at 0,0,0
glTranslatef(2,0,0); // A dot at 0,0,0
glTranslatef(1.41,0,0); // A dot at 0,0,0
glRotatef(90,0,0,1); // A dot at 0,0,0
glTranslatef(1,1,0); // A dot at 0,0,0
glRotate(45,0,0,1); // A dot at 0,0,0

Unchanged
A hierarchy

- Draw in W
- W->L1; Draw in L1
- L1->L2; Draw in L2
- L2->L3; Draw in L3
- L3->L2;
- L2->L4; Draw in L4
A hierarchy

- Draw in W
- W->L1; Draw in L1
- L1->L2; Draw in L2
- L2->L3; Draw in L3
- L3->L2;
- L2->L4; Draw in L4
- L4->L2
- L2->L1
- L1->L5; Draw in L5
Summary

• The skeleton model can be defined as a tree.

• Rendering can be done by depth-first transversal.

• Matrix and coordinate system has the same idea: the matrix gives

• You can use inverse transformation (not a good way) to rewind:
  • such as L3->L2, L2->L1...
Matrix Stack

//Transformation+Draw
glPushMatrix();
//Transformation+Draw
glPushMatrix();
//Transformation+Draw
glPushMatrix();
//Transformation+Draw
glPushMatrix();
//Transformation+Draw
glPopMatrix();
Matrix Stack

Instead of doing inverse transformation, we can easily rewind using the matrix stack.

```
// Starts with W
// Do W->L1
glPushMatrix();  // Back up L1
// Do L1->L2
glPushMatrix();  // Back up L2
// Do L2->L3
glPushMatrix();  // Back up L3
// Do L2->L4
glPopMatrix();   // Restore L2
// Do L2->L4
glPopMatrix();   // Restore L1
// Do L1->L5
glPopMatrix();   // Restore L1
// Done
```

Depth-First Traversal
Motion Capture
Performance Capture
The **Biovision Hierarchy (BVH)** file format was developed by Biovision, a defunct motion capture company.

- It stores motion capture data.
- It’s a text file.
- It matches with OpenGL well. (That is why you will learn how to use it in Lab2.)
- It has two parts: hierarchy and data.
Part 1: Hierarchy

• The hierarchy is a joint tree.
• Each joint has an offset and a channel list.

• Offset is a **bone** vector (defined in \( L_1 \)) from Joint 1 to Joint 2.
• Let \( L_{12} \) be a shifted \( L_1 \) at Joint 2.
• The channel list defines a sequence of transformations from \( L_{12} \) to \( L_2 \).
Part 1: Hierarchy

- The channel list uses nine deformations:
  - Xposition, Yposition, Zposition
  - Xrotation, Yrotation, Zrotation
  - Xscale, Yscale, Zscale

- The list only tells us transformation types and their orders. The order is the same as in OpenGL.

- To know the actual transformation, we need to read the data.
An example

**Chest Joint and its local coordinate system**

```
JOINT chest
{
    OFFSET  0.096536  3.475309  -0.289609
    CHANNELS 3 Xrotation  Zrotation  Yrotation
}
```

**Neck Joint and its local coordinate system**

```
JOINT neck
{
    OFFSET  -0.096536  13.901242  -2.027265
    CHANNELS 3 Xrotation  Zrotation  Yrotation
}
```

Neck’s offset from chest (like a bone)

```
JOINT head
{
    OFFSET  -0.166775  1.448045  0.482682
    CHANNELS 3 Xrotation  Zrotation  Yrotation
}
```

Channel list:

Transformation from chest coordinate system to neck coordinate system
Part 2: Data

- The data part stores a number of frames.
- Each frame is a list of variables.
- The model is animated, when each frame uses variables.
How to use BVH.h

• BVH.h does most things for you.
• Your job is to complete the render function.

```cpp
class BVH
{
public:
    BVH_NODE root;
    ...
    void Read_File(...);
    void Set_Data(frame_id);
    void Forward();
    void Backward();
    void Render();
}
```
```
class BVH_NODE
{
public:
    char name[1024];
    float offset[3];
    BVH_NODE* children;
    int children_number;
    float *tx, *ty, *tz;
    float *rx, *ry, *rz;
    float *sx, *sy, *sz;
    int data_order[9];
}
```
The Data Order

In data_order:
1 means tx; 2 means ty; 3 means tz
4 means rx; 5 means ry; 6 means rz
7 means sx; 8 means sy; 9 means sz
0 means end

- Translate *tz
- Translate *ty
- Scale *sx
- Scale *sz
- Rotate *ry
- end
The Camera View

```
    gluLookAt(eye_x, eye_y, eye_z, center_x, center_y,
              center_z, up_x, up_y, up_z)
```

• To change the camera view
  • We can update the `gluLookAt` function (which is not very easy...).
  • Or, we can combine `gluLookAt` with transformation functions. (Check Lab2 for details.)