

Outline

- Object Transformations
- Moving Things Around
- Hierarchical Modeling
 - Implementing Hierarchies

World and Object Frames

CSC471

- we can express objects in its own object frame and then transform the points to the world frame.

CSC471

World and Object Frames

CSC471

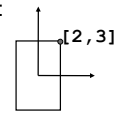
- The relationship between the object frame and the world frame is represented by the matrix \mathbf{O} :
- We may have multiple matrices \mathbf{O}_i , one for each object.
- \mathbf{O}_i relates the world frame to object i 's coordinate system.
- This means we can express the object in its own object frame and then transform the points to the world frame.

CSC471

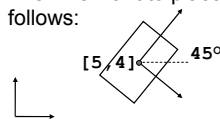
OpenGL Example

CSC471

- Suppose we want to draw the following 2D rectangle within its object frame:



- Then we want to place it in the world frame as follows:



CSC471

Drawing the Object

CSC471

- Here is the code to draw the rectangle in its frame:



- Note how easy it is to draw objects using their local object frame.
- The relationship between the object frame and the world frame is represented by the matrix \mathbf{O}_i :

CSC471

Transforming the Object Frame

CSC471

- \mathbf{O}_i encompasses the following transformations:
 - 1. A clockwise rotation about the origin by 45° .
 - 2. A translation by $[5, 4]$.
- The rotation matrix is:

- The translation matrix is:

CSC471

Transforming the Object Frame

- The combined transformation matrix is:
- Note the order of transformations.
 - Read it right-to-left.

Transforming the Object Frame

- The OpenGL code for this transformation is:

```
void
setupObj1(void) {
    glTranslatef( 5.0, 4.0, 0.0 );
    glRotatef( -45.0, 0.0, 0.0, 1.0 );
}
```

- To go back to the previous CTM, we wrap this function in `glPushMatrix()` and `glPopMatrix()` calls.

Drawing the Scene

- To draw the scene, we use the following OpenGL

```
code: displayCB(void) {
    glClear( GL_COLOR_BUFFER_BIT ); // Clear the image

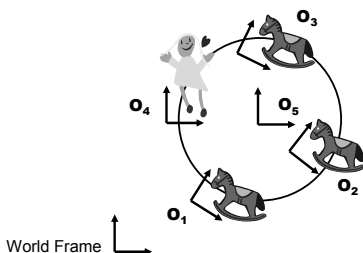
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    // Set up the view
    setupView();
    // Now we are in the world frame
    glPushMatrix();
    setupObj1();
    // Now we are in the object1 frame
    drawObj1();
    glPopMatrix();
    // Back in the world frame again
    glutSwapBuffers();
}
```

Outline

Object Transformations
 Moving Things Around
 Hierarchical Modeling
 Implementing Hierarchies

Example: Merry-go-Round

- O_i describe how parts are placed with respect to the world frame.



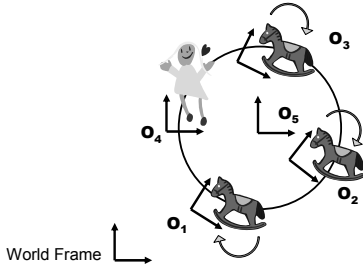
Moving Things Around

- Updates on an objects position and orientation is done by appropriately updating its frame, which is represented by its object matrix O_i .
- When we move things around, we always have to say with respect to which frame we are moving an object.
- We can move things with respect to:
 - The object's own frame.
 - The world frame.
 - Another frame.

Local Motion

CSC471

- Rotate horses with respect to their own frame.



CSC471

Local Motion

CSC471

- This is the easiest motion to accomplish.
- The object will rotate and translate depending on its current orientation.
- If Q is the matrix representing the rotation/translation, then:
 - That means the new object matrix O_i is:
 - We call Q a modeling transformation.
 - Apply Q to object first!!

CSC471

OpenGL

CSC471

- Apply modeling transformation before object transformation.

```
void moveHorse(void) {
    glTranslatef( Th1[0], Th1[1], 0.0 );
    glRotatef( Ah1, 0.0, 0.0, 1.0 );
}

void moveHorse(void) {
    glRotatef( Ah, 0.0, 0.0, 1.0 );
}

void display(void) {
    ...
    // Save CTM to stack
    glPushMatrix();
    setupHorse();
    moveHorse();
    glColor3f(1.0, 0.0, 0.0);
    drawHorse();
    glPopMatrix();
    ...
}
```

CSC471

Animation

CSC471

- We can simply change the variables to the OpenGL calls over time.
- For example, we can change the angle Ah by incrementing it every time step.
- When and where do we change the variables?
- When do we call `glutSwapBuffers()` ?

CSC471

Animation 1

CSC471

- We can update the variables in the idle callback:

```
void displayCB() {
    ...
    glutSwapBuffers();
}

void idleCB() {
    Ah += ROTATION_INC;
    glutPostRedisplay();
}
```

Carousel1

CSC471

Animation 2

CSC471

- We can call the GLUT timerCB at specified intervals:

```
void timerCB() {
    Ah += ROTATION_INC;
    glutTimerFunc(1000/24, timerCB, 0);
    glutPostRedisplay();
}

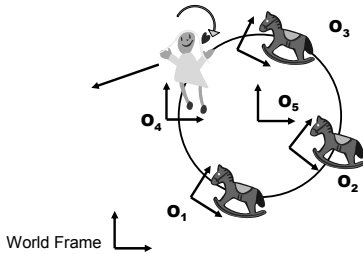
void displayCB() {
    ...
    glutSwapBuffers();
}
```

Carousel2

CSC471

Global Motion

- Translate and rotate girl with respect to world frame.



Global Motion

- Direction of translation and rotation relative to the world frame.
- If Q is the matrix representing the motion, then:
 - And so the new object matrix O_i is:
 - We call Q an instance transformation.
 - Set up object then apply transform

OpenGL

- Apply instance transformation after object transformation

```

void moveGirl(void) {
    glTranslatef( -5.0, 5.0, 0.0 );
    glRotatef( 30, 0.0, 0.0, 1.0 );
}

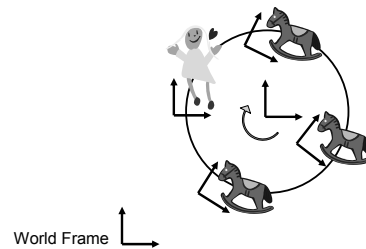
void moveGirl(void) {
    glTranslatef( Tobj[0], Tobj[1], 0.0 );
    glRotatef( Aobj, 0.0, 0.0, 1.0 );
}

void display(void) {
    ...
    // Save CTM to stack
    glPushMatrix();
    moveGirl();
    setupGirl();
    glColor3f(1.0, 0.0, 0.0);
    drawGirl();
    glPopMatrix();
    ...
  
```

Carousel3

Relative Motion

- Rotate object frames with respect to carousel frame.



Relative Motion

- Often we want to transform frame F_o by some matrix R with respect to some auxiliary frame G_o .
- Suppose the two frames are related by:
 - The transformed frame can then be expressed as:
 - We can use Push / Pop matrix calls and hierarchical modeling to achieve this effect.

Relative Motion

- Often we want to transform frame F_o by some matrix R with respect to some auxiliary frame G_o .
- We can use Push / Pop matrix calls and hierarchical modeling to achieve this effect.
 - Outer most transform call executed first...

OpenGL

- Use nested `glPushMatrix()` / `glPopMatrix()` calls:

```
void display(void) {
...
  glPushMatrix();
  setupCarousel();
  moveCarousel();
  glColor3f(0.0, 1.0, 1.0);
  drawAxis();

  // Draw horses
  glPushMatrix();
  setupHorse1();
  moveHorse1();
  glColor3f(1.0, 0.0, 0.0);
  drawHorse1();
  glPopMatrix();
...
  glPopMatrix();
...
}
```

Carousel4

Outline

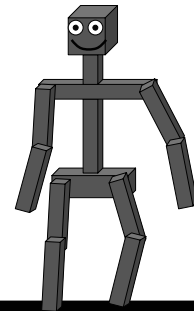
- Object Transformations
- Moving Things Around
- Hierarchical Modeling
 - Implementing Hierarchies

Hierarchical Modeling

- Many objects are naturally modeled hierarchically.
- Example: A tree.
 - The trunk starts out at the ground.
 - The large branches start out from the trunk.
 - The small branches start out from the large branches.
 - The leaves start out from the small branches.
- It is best to describe each successive "child" frame (trunk, big branch, small branch, leaf) with respect to its previous parent frame.

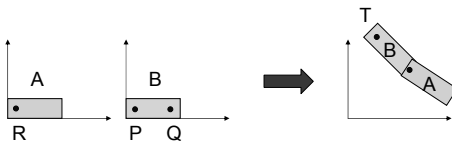
Hierarchical Modeling

- A lesson in stick person anatomy.



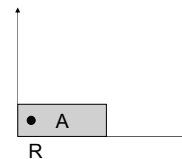
Making an Articulated Arm

- A minimal 2D jointed object:
 - Two pieces, *A* ("forearm") and *B* ("upper arm").
 - Attach point *Q* on *B* to point *R* on *A* ("elbow").



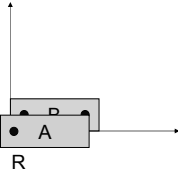
How To Do It

- Draw *A* and *B* in their local object frame.
- B* is hiding behind *A*.



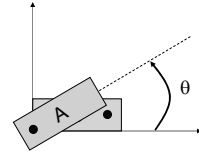
How To Do It

- 1. Translate A by $(-R_x, -R_y)$



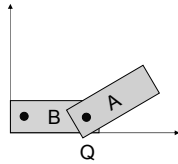
How To Do It

- 2. Rotate A by $+\theta$ (the "elbow" angle).



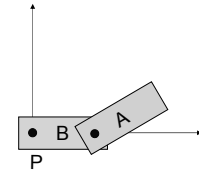
How To Do It

- 3. Translate A by (Q_x, Q_y) to align the points R and Q.



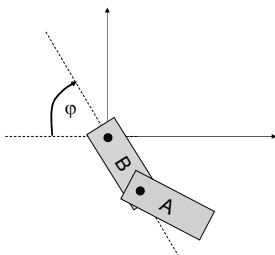
How To Do It

- 4. Translate A & B by $(-P_x, -P_y)$.



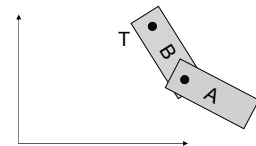
How To Do It

- 5. Rotate CW by ϕ (the "shoulder" angle).



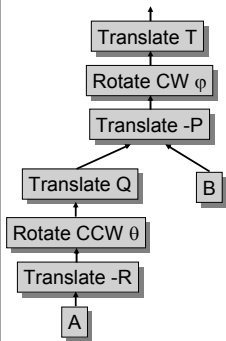
How To Do It

- 6. Translate by (T_x, T_y) to the final shoulder position T.



Transformation Hierarchies

CSC471

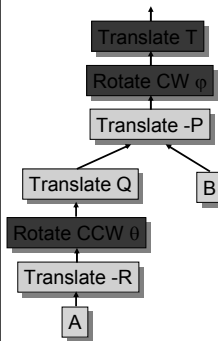


- This is the build-an-arm sequence, represented as a tree.
- Interpretation:
 - Leaves are geometric primitives.
 - Non-leaves are transformations.
 - Transformations apply to everything under them — start at the bottom and work your way up.

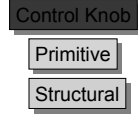
CSC471

Transformation Hierarchies

CSC471



- You can build a wide range of models this way.



CSC471

What have we done?

CSC471

- Seems more complicated than just translating and rotating each piece separately.
- We have the following control knobs:
 - T: shoulder position (point at which P winds up.)
 - φ : shoulder angle (A and B rotate together about P.)
 - θ : elbow angle (A rotates about R, which stays attached to P.)
- φ , θ , and T are parameters of the model.
 - Changing them wiggles the arm.
- P, Q, and R are structural constants.
 - Changing them dismembers the arm.

CSC471

The Right Control Knobs

CSC471

- These are a set of “control knobs” (parameters) that make it easy to move our stick person.
- This kind of control is convenient for static models, and vital for animation.
- We also need hierarchical modeling.
- Without it, the model falls apart as soon as you change something.



CSC471

Outline

Moving Things Around
 Hierarchical Modeling
 Implementing Hierarchies

Implementing Hierarchies

CSC471

- Building block: a matrix stack that you can push / pop.
- A simple recursive algorithm that descends your model tree, doing transformations, pushing, popping, and drawing.
- OpenGL's matrix operations are tailored to doing this.

CSC471

The Algorithm

CSC471

- Using the stack operations a very simple recursive descent algorithm does the trick:
 - Initially, load the identity matrix.
 - At each non-leaf node:
 - Push the stack.
 - Perform transformation(s).
 - Recursively draw the children.
 - Pop and return.
 - At a leaf nodes, draw the geometric primitive(s).

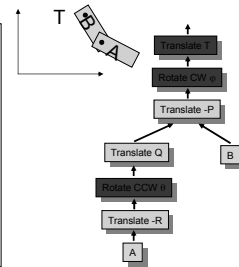
CSC471

Two Link Arm, Revisited

CSC471

- OpenGL transformations:

```
glPushMatrix();
glTranslatef(Tx,Ty,0);
glRotatef(-φ,0,0,1);
glTranslatef(-Px,-Py,0);
glPushMatrix();
glTranslatef(Qx,Qy,0);
glRotatef(θ,0,0,1);
glTranslatef(-Rx,-Ry,0);
Draw(A);
glPopMatrix();
Draw(B);
glPopMatrix();
```



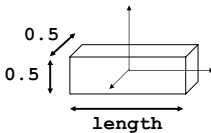
CSC471

Two Link Arm, Revisited

CSC471

- OpenGL box drawing:

```
void drawBox(void)
{
    // Draw box
    glColor3f(1.0, 1.0, 1.0);
    glScalef(length, 0.5, 0.5);
    glutWireCube(1.0);
}
```



CSC471

Two Link Arm, Revisited

CSC471

```
void drawArm(int Tx, int Ty, int Tz,
            double shoulderAngle,
            double elbowAngle)
{
    // Hierarchical Transformations
    glPushMatrix();
    glTranslatef(Tx,Ty,Tz);
    glRotatef(360.0-shoulderAngle,0,0,1);
    glTranslatef(-Px,-Py,-Pz);
    glPushMatrix();
    // Draw elbow
    glTranslatef(Qx,Qy,Qz);
    glRotatef(elbowAngle,0,0,1);
    glTranslatef(-Rx,-Ry,-Rz);
    drawBox();
    glPopMatrix();
    // Draw biceps
    drawBox();
    glPopMatrix();
}
```

CSC471

Two Link Arm, Revisited

CSC471

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    // Reset Modelview Matrix
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Viewing Transformations
    setupView();

    // Draw axis
    drawAxis();
    // Draw object
    drawArm(1, 1, 0, // Shoulder pos
           60, // Shoulder angle (CW)
           10); // Elbow angle (CCW)
    glutSwapBuffer();
}
```

Arm

CSC471

A Schematic Humanoid

CSC471



- The root can be anywhere.
- We chose the hip.
- You get control knobs for each joint angle, plus global position and orientation.
- A realistic human would be *much* more complex.
 - But stick persons are also people...

CSC471