Hierarchical Modeling: Tree of Transformations, Display Lists and Functions, Matrix and Attribute Stacks,

Modeling complex objects/motion

- Decompose object hierarchically into parts
- Model each part in local frame, and use affine transformations to position it at its proper place in the part above it in the hierarchy
The pipeline conceptually

- Construct shapes from geometric primitives
- Arrange the objects in three dimensional space and select the desired vantage point for viewing the composed scene.
- Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, or obtained by pasting a texture onto the objects.
- Convert the mathematical description of objects and their associated color information to pixels on the screen.
- During these stages, an API might perform other operations, such as eliminating parts of objects that are hidden by other objects.

The Rendering

![Image of a 3D Buddha model]
World Coordinates and Model Coordinates
Viewing the World

Viewing Transformation
MVT, Lights, Material, Shading, ...
Rendering

Coordinate Systems
- Object/Model
- World
- Camera/eye
- Image/Normalized View Volume
- Screen, Pixel
Current Transformation

- $\text{CTM} = \text{PT} \times \text{MVT}$.
- MVT and PT are usually part of the current state.
- Matrix Stacks in OpenGL.

Possible approaches

- Symbols and instancing, instancing transforms: can be represented by a table.
- Hierarchical modeling. Scene graphs (SG).
  - (Scene) Trees
  - DAGs
Instancing in OpenGL

- In OpenGL, instancing is created by modifying the model-view matrix:

```c
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glPushMatrix();
glTranslatef( ... );
glRotatef( ... );
glScalef( ... );
symbol();
glPopMatrix( );
```

Symbols & Instances
Hierarchical Modeling

DAGs are natural in many applications, e.g., modeling polygonal meshes where many edges and vertices are shared by higher level objects.
Hierarchical Modeling
using Trees

- Primitives that can be drawn directly, at leaves
- Internal nodes become instances
- Transformations that position instances of objects label the edges
- Traverse in pre-order (visit parent, visit children left to right).
  - Stack based implementation
  - Model independent traversal algorithm
Example: simple 2D drawing

Hierarchical modeling: tree of transformations

T1.T2.T5.T7.T9(LINE)

Final picture, obtained by traversing in pre-order the tree
Compress the tree

- If an edge \((A,B)\) is labeled with identity transformation, remove it, and move any drawings from node B into A.
- If edges \((A,B)\) and \((A,C)\) have the same labels, merge the nodes B and C.
- Example: in the previous tree T8 will be removed, and the unit square moved to the parent node.

From the tree model to OpenGL code

- Start from the root, traverse tree in pre-order.
- When visiting a node, if there is a primitive at the node that can be drawn, draw it.
- If going down from a node which has more than one unvisited child, store(push) CTM and attributes before continuing down.
- If coming back up an edge to a node which has unvisited children, pop CTM and attributes.
Hierarchical Modeling Hofstra University

Stack implementation of the compressed tree

```c
glScalef(1.0, -1.0, 1.0);        // T1
glPushMatrix();
glTranslatef(1.5, 3.0, 0.0);    // T2
glScalef(1.0, 2.0, 1.0);        // T5
glBegin(GL_LINE_LOOP);   // BOX
   glVertex3f(0.0, 0.0, 0.0);
   glVertex3f(1.0, 0.0, 0.0);
   glVertex3f(1.0, 1.0, 0.0);
   glVertex3f(0.0, 1.0, 0.0);
glEnd();

glTranslatef(0.5, 0.0, 0.0);    // T7
glRotatef(90.0, 0.0, 0.0, 1.0); // T9
glBegin(GL_LINES);             // LINE
   glVertex3f(0.0, 0.0, 0.0);
   glVertex3f(1.0, 0.0, 0.0);
   glVertex3f(1.0, 1.0, 0.0);
   glVertex3f(0.0, 1.0, 0.0);
glEnd();

glPopMatrix();

glPushMatrix();
glTranslatef(2.0, 1.0, 0.0);   // T3
glScalef(0.5, 0.5, 1.0);       // T6
drawUnitCircle(NUM_SLICES);

glPopMatrix();

glScalef(4.0, 5.0, 1.0);       // T4
glBegin(GL_LINE_LOOP);   // BOX
   glVertex3f(0.0, 0.0, 0.0);
   glVertex3f(1.0, 0.0, 0.0);
   glVertex3f(1.0, 1.0, 0.0);
   glVertex3f(0.0, 1.0, 0.0);
glEnd();

glFlush();
```

Display Lists

- Need to draw the display at a rate sufficient to avoid noticeable flicker
**Display Processor**

- **Display processor** – special purpose, limited instruction set, oriented toward drawing graphics primitives
- **Display memory** holds lists of instructions called display lists
- Display list executed repeatedly at a rate sufficient to avoid flicker

**Display Modes**

- **Immediate Mode** – as soon as the program executes a statement that defines a primitive, it is sent to the server
- **Retained Mode** – define the object once, then put its description in a display list. Redisplayed with a simple function call
- Reduced network traffic
OpenGL Display Lists

```c
#define BOX 1
glNewList(BOX, GL_COMPILE);
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(-1.0, -1.0);
    glVertex2f( 1.0, -1.0);
    glVertex2f( 1.0,  1.0);
    glVertex2f(-1.0,  1.0);
    glEnd();
    glEndList();
gCallList(BOX);
```

Send list to server but don't display contents

GL_COMPILE_AND_EXECUTE

This can change the original state and leave it altered

Definition and Execution

```c
#define BOX 1
glNewList(BOX, GL_COMPILE);
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glPushMatrix();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(-1.0, -1.0);
    glVertex2f( 1.0, -1.0);
    glVertex2f( 1.0,  1.0);
    glVertex2f(-1.0,  1.0);
    glEnd();
    glPopAttrib();
    glPopMatrix();
    glEndList();
gCallList(BOX);
```

Push and pop attributes and coordinates to the stacks
Stack implementation of the tree of transformations

```c
glScalef(1.0,-1.0,1.0);

glPushMatrix();
glTranslatef(1.5, 3.0, 0.0);
glScalef(1.0, 2.0, 1.0);

glCallList(BOX);
glTranslatef(0.5,0.0, 0.0);
glRotatef(90.0,0.0,0.0,1.0);
glBegin(GL_LINES);
  glVertex3f(0.0,0.0,0.0);
  glVertex3f(1.0,0.0,0.0);
glEnd();

glPopMatrix();


glPushMatrix();
glTranslatef(2.0,1.0,0.0);
glScalef(0.5, 0.5, 1.0);
drawUnitCircle(NUM_SLICES);
glPopMatrix();
glScalef(4.0,5.0,1.0);
glCallList(BOX);

glFlush();
```

Example: cart with two wheels
Example: wheel

Model a red wheel of radius WHEEL_RAD, consisting of a circle and two crossed spikes. Primitive components are unit circle and crossed orthogonal spikes of length 2.

```gl
glNewList(WHEEL, GL_COMPILE); // red wheel with WHEEL_RAD
    glPushMatrix();
    glPushAttrib(GL_COLOR);
    glColor3f(1.0, 0.0, 0.0); // red
    glScalef(WHEEL_RAD, WHEEL_RAD, 1.0); // scale to wheel radius
    glBegin(GL_LINES); // cross, each bar length 2
        glVertex2f(-1.0, 0.0);
        glVertex2f(1.0, 0.0);
        glVertex2f(0.0, -1.0);
        glVertex2f(0.0, 1.0);
    glEnd();
    drawUnitCircle(NUM_SLICES); // unit circle
    glPopAttrib();
    glPopMatrix();
    glEndList();
```

Example: rod with 2 fixed wheels

Model the cart consisting of two red wheels of radius WHEEL_RAD, positioned at the ends of a green rod of length 2*HALF_AXIS_LENGTH. The two wheels are rotated at different angles.

```gl
glNewList(CART, GL_COMPILE);
    glPushMatrix();
    glPushAttrib(GL_COLOR);
    glColor3f(0.0, 1.0, 0.0); // green
    glBegin(GL_LINES); // rod
        glVertex2f(-HALF_AXIS_LENGTH, 0.0);
        glVertex2f(HALF_AXIS_LENGTH, 0.0);
    glEnd();
    glPushAttrib();
    glPushMatrix();
        glTranslatef(-HALF_AXIS_LENGTH, 0.0, 0.0); // position front WHEEL at axis
        glRotatef(TH, 0.0, 0.0, 1.0); // rotate a WHEEL by TH
        glCallList(WHEEL);
    glPopMatrix();
    glTranslatef(HALF_AXIS_LENGTH, 0.0, 0.0); // position back WHEEL at axis
    glRotatef(TH+INCR, 0.0, 0.0, 1.0); // rotate a WHEEL by TH+INCR
    glCallList(WHEEL);
    glPopAttrib();
    glPopMatrix();
    glEndList();
```
Example: cart with 2 wheels

```c
void display()
{
    glClearColor(1.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-225.0, 225.0, -225.0, 225.0);
    glMatrixMode(GL_MODELVIEW);

    defineParts(); // WHEEL and CART are display lists defined here,
}

void myinit()
{
    glClearColor(1.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-225.0, 225.0, -225.0, 225.0);
    glMatrixMode(GL_MODELVIEW);

    defineParts(); // WHEEL and CART are display lists defined here,
}

main() .... As usual, negotiate with Windows Sys, register callbacks, run loop
        to process events
```

Example: spin the wheels

- Limitations of display lists
- Use of functions in drawing instances of objects at tree nodes
- Continuously keep changing the value of rotation angle for turning a wheel
- Need of double buffering
Example: spin the wheels

```c
void draw_cart()
{
    glPushMatrix();
    glPushAttrib(GL_COLOR);
    glColor3f(0.0,1.0,0.0); // draw axis in green
    glBegin(GL_LINES);
        glVertex2f(-HALF_AXIS_LENGTH,0.0);
        glVertex2f(HALF_AXIS_LENGTH,0.0);
    glEnd();
    glPushMatrix();
    glTranslatef(-HALF_AXIS_LENGTH,0.0, 0.0); // position front wheel at axis, T1
    glRotatef(theta,0.0,0.0,1.0);        // spin front WHEEL by theta
    glCallList(WHEEL); // draw wheel
    glPopMatrix();
    glTranslatef(HALF_AXIS_LENGTH,0.0, 0.0);  // position back wheel at axis, T2
    glRotatef(theta, 0.0,0.0,1.0);           // spin back WHEEL by theta
    glCallList(WHEEL); // draw wheel
    glPopAttrib();
    glPopMatrix();
}
```

Open GL interactive program structure

```c
int main(int argc , char** argv )
{
    glutInit (&argc ,argv );
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow ("generic example");
    myinit ();
    glutReshapeFunc (myReshape);
    glutMouseFunc (mouse);
    glutMotionFunc (do_on_motion);
    glutIdleFunction (idle);
    glutDisplayFunc (display);

    glutMainLoop();
}
```
void idle(void)                        // IDLE callback function updates
{                                               // rotation angle that spins a wheel
    theta += INCR_ANGLE;
    glutPostRedisplay();
}
void mouseMotion(int x, int y)          // MOTION callback function updates
{                                               // translation vector that moves whole
    transx = x; transy = y;
    glutPostRedisplay();
}
void display()                         // DISPLAY callback draws whole object
{                                               // in back buffer then swaps buffers
    glClear(GL_COLOR_BUFFER_BIT);
    // translate whole object axis with spinning wheels
    glTranslatef( (float)transx, (float)(WINDOWH-transy), 0.0);
    draw_cart();
    glutSwapBuffers();                        // SWAP BUFFERS
}

Example: spin the wheels, 4, the Idle Function

int main(…)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB); // double buffering
    glutDisplayFunc(display);
    glutMotionFunc(mouseMotion);
    glutIdleFunc(idle);
    glutMainLoop();
    …
}
HW: Model hierarchically a 3D scene consisting of a circular road, a car, at least 2 trees along the road. Each component has a different color than the rest.

- Allow for interactively changing the camera position, allowing the camera to rotate around the horizontal and vertical axis of the coordinate system attached to the scene, and also for interactively choosing an option to move the car forward or backward
- A car has a rectangular body and 4 wheels
- A tree has a trunk (cylinder) and a crown (sphere)
- The road is a circular loop
- Model each object (object part) in its own local coordinate system; use transformations to position all objects in the scene.
- Trees of transformations should help in modeling the individual objects (tree, car), and also in putting the whole scene together.
- Use functions or display lists to model the components
- HW is due in 2 weeks. Pace yourself well: complete the modeling in a week, so the week after you could deal with the interaction and the animation. Model and test each object individually, first.

Example: model circular road

```c
// ROAD_RAD1, ROAD_RAD2, ROAD_SL, ROAD_RINGS defined globally

void draw_road()
{
    GLUquadric* q;    // dynamic array to hold vertices of road
    q = gluNewQuadric(); // creates and returns a ptr to new quadric object

    glPushMatrix();
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glColor3f(0.5,0.5,0.5); // road is gray
    gluDisk(q, ROAD_RAD1, ROAD_RAD2, ROAD_SL, ROAD_RINGS);

    glPopAttrib();
    glPopMatrix();
}
```

Hierarchical Modeling  Hofstra University  39
Example: model circular road

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    // glRotatef(45.0, 0.0, 0.0, 0.0);
    // gluLookAt(-1.0, -1.0, -1.0, 0.0, 0.0, 1.0);
    draw_road();
    glFlush();
}