Objectives

- Examine the limitations of linear modeling
  - Symbols and instances
- Introduce hierarchical models
  - Articulated models
  - Robots
- Introduce Tree and DAG models
- Build a tree-structured model of a humanoid figure
- Examine various traversal strategies
- Build a generalized tree-model structure that is independent of the particular model

Instance Transformation

- Start with a prototype object (a symbol)
- Each appearance of the object in the model is an instance
  - Must scale, orient, position
  - Defines instance transformation

Symbol-Instance Table

Can store a model by assigning a number to each symbol and storing the parameters for the instance transformation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Scale</th>
<th>Rotate</th>
<th>Translate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9, 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>0, 0, 0</td>
</tr>
</tbody>
</table>

Relationships in Car Model

- Symbol-instance table does not show relationships between parts of model
- Consider model of car
  - Chassis + 4 identical wheels
  - Two symbols
- Rate of forward motion determined by rotational speed of wheels

Structure Through Function Calls

```python
def car(speed):
    chassis()
    wheel(right_front);
    wheel(left_front);
    wheel(right_rear);
    wheel(left_rear);
```

- Fails to show relationships well
- Look at problem using a graph
Graphs

• Set of nodes and edges (links)
• Edge connects a pair of nodes
  • Directed or undirected
• Cycle: directed path that is a loop

Tree

• Graph in which each node (except the root) has exactly one parent node
  • May have multiple children
• Leaf or terminal node: no children

Tree Model of Car

DAG Model

• If we use the fact that all the wheels are identical, we get a directed acyclic graph
  • Not much different than dealing with a tree

Modeling with Trees

• Must decide what information to place in nodes and what to put in edges
  • Nodes
    • What to draw
    • Pointers to children
  • Edges
    • May have information on incremental changes to transformation matrices (can also store in nodes)

Robot Arm

• robot arm
  • parts in their own coordinate systems
Articulated Models

- Robot arm is an example of an articulated model
- Parts connected at joints
- Can specify state of model by giving all joint angles

Relationships in Robot Arm

- Base rotates independently
  - Single angle determines position
- Lower arm attached to base
  - Its position depends on rotation of base
  - Must also translate relative to base and rotate about connecting joint
- Upper arm attached to lower arm
  - Its position depends on both base and lower arm
  - Must translate relative to lower arm and rotate about joint connecting to lower arm

Required Matrices

- Rotation of base: \( R_b \)
  - Apply \( M = R_b \) to base
- Translate lower arm relative to base: \( T_{lu} \)
- Rotate lower arm around joint: \( R_{lu} \)
  - Apply \( M = R_l T_{lu} R_{lu} \) to lower arm
- Translate upper arm relative to upper arm: \( T_{uu} \)
- Rotate upper arm around joint: \( R_{uu} \)
  - Apply \( M = R_u T_{uu} R_{uu} T_{uu} R_{uu} \) to upper arm

OpenGL Code for Robot (Immediate mode)

```c
mat4 ctm;
robot_arm()
{
    ctm = RotateY(theta);
    base();
    ctm *= Translate(0.0, h1, 0.0);
    ctm *= RotateZ(phi);
    lower_arm();
    ctm *= Translate(0.0, h2, 0.0);
    ctm *= RotateZ(psi);
    upper_arm();
}
```

Tree Model of Robot

- Note code shows relationships between parts of model
  - Can change “look” of parts easily without altering relationships
- Simple example of tree model
- Want a general node structure for nodes

Possible Node Structure

```
+-----+       +-----+
|     |       |     |
|     |       +-----+
| Base|       +-----+
|     |       |     |
+-----+       +-----+
      |     |
      +-----+
      |     |
      |     |
      +-----+
        |   |
        +-----+
        |     |
        |     |
        +-----+
```

- Code for drawing part or pointer to drawing function
- Linked list of pointers to children
- Matrix relating node to parent
Generalizations

- Need to deal with multiple children
  - How do we represent a more general tree?
  - How do we traverse such a data structure?
- Animation
  - How to use dynamically?
  - Can we create and delete nodes during execution?

Humanoid Figure

Building the Model

- Can build a simple implementation using quadrics: ellipsoids and cylinders
- Access parts through functions
  - torso()
  - left_upper_arm()
- Matrices describe position of node with respect to its parent
  - $M_{lla}$ positions left lower leg with respect to left upper arm

Display and Traversal

- The position of the figure is determined by 11 joint angles (two for the head and one for each other part)
- Display of the tree requires a graph traversal
  - Visit each node once
  - Display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation

Transformation Matrices

- There are 10 relevant matrices
  - $M$ positions and orients entire figure through the torso which is the root node
  - $M_{h}$ positions head with respect to torso
  - $M_{lla}$, $M_{rla}$, $M_{lll}$, $M_{rll}$ position lower parts of limbs with respect to corresponding upper limbs
Stack-based Traversal

- Set model-view matrix to $M$ and draw torso
- Set model-view matrix to $MM_h$ and draw head
- For left-upper arm need $MM_{lua}$ and so on
  - We could compute $MM_{lua}$ from scratch or using an inverse matrix
  - We could use a matrix stack to store $M$ and other matrices as we traverse the tree (easy with < 3.1)

Traversal Code

```c
figure()

PushMatrix()
  torso();
  Rotate(...);
  head();
 PopMatrix();

PushMatrix();
  Translate(...);
  Rotate(...);
  left_upper_arm();
 PopMatrix();

MarshalAs() {  
  save present model-view matrix  
  PushMatrix();  
  torso();  
  update model-view matrix for head  
  Rotate(...);  
  head();  
  recover original model-view matrix  
  PopMatrix();  
  PushMatrix();  
  Translate(...);  
  Rotate(...);  
  left_upper_arm();  
  update model-view matrix for left upper arm  
  PopMatrix();  
  recover and save original model-view matrix again  
  PushMatrix();  
  rest of code
```

Analysis

- The code describes a particular tree and a particular traversal strategy
- Can we develop a more general approach?
- Note that the sample code does not include state changes, such as changes to colors
- May also want to use `glPushAttrib` and `glPopAttrib` to protect against unexpected state changes affecting later parts of the code

General Tree Data Structure

- Need a data structure to represent tree and an algorithm to traverse the tree
- We will use a left-child right sibling structure
  - Uses linked lists
  - Each node in data structure is two pointers
    - Left: next node
    - Right: linked list of children

Tree node Structure

- At each node we need to store
  - Pointer to sibling
  - Pointer to child
  - Pointer to a function that draws the object represented by the node
  - Homogeneous coordinate matrix to multiply on the right of the current model-view matrix
    - Represents changes going from parent to node
    - In OpenGL this matrix is a 1D array storing matrix by columns

Left-Child Right-Sibling Tree
**C Definition of treenode**

```c
typedef struct treenode
{
    mat4 m;    // GLfloat m[16];
    void (*f)();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
```

**Defining the torso node**

```c
treenode torso_node, head_node, lua_node, ...;
torso_node.m = RotateY(theta[0]);
torso_node.f = torso;
torso_node.sibling = NULL;
torso_node.child = &head_node;
head_node.m = translate(0.0, TORSO_HEIGHT+0.5*HEAD_HEIGHT, 0.0)*
    RotateX(theta[1])*RotEtY(theta[2]);
head_node.f = head;
head_node.sibling = &lua_node;
head_node.child = NULL;
```

**Notes**

- The position of figure is determined by 11 joint angles stored in `theta[11]`
- Animate by changing the angles and redisplaying
- We form the required matrices using `Rotate` and `Translate`
  - More efficient than software
  - Because the matrix is formed in model-view matrix, we may want to first push original model-view matrix on matrix stack

**Preorder Traversal**

```c
void traverse(treenode* root)
{
    if(root==NULL) return;
    mvstack.push(model_view);
    model_view = model_view*root->m;
    root->f();
    if(root->child!=NULL)
        traverse(root->child);
    model_view = mvstack.pop();
    if(root->sibling!=NULL)
        traverse(root->sibling);
}
```

**Dynamic Trees**

- If we use pointers, the structure can be dynamic
  ```c
typedef treenode *tree_ptr;
tree_ptr torso_ptr;
torso_ptr = malloc(sizeof(treenode));
```
  - Definition of nodes and traversal are essentially the same as before but we can add and delete nodes during execution
GRAPHICAL OBJECTS AND SCENE GRAPHS

Objectives

- Introduce graphical objects
- Generalize the notion of objects to include lights, cameras, attributes
- Introduce scene graphs

Limitations of Immediate Mode Graphics

- When we define a geometric object in an application, upon execution of the code the object is passed through the pipeline.
- It then disappears from the graphical system.
- To redraw the object, either changed or the same, we must reexecute the code.
- Display lists provide only a partial solution to this problem.

OpenGL and Objects

- OpenGL lacks an object orientation.
- Consider, for example, a green sphere.
- We can model the sphere with polygons or use OpenGL quadrics.
- Its color is determined by the OpenGL state and is not a property of the object.
- Defies our notion of a physical object.
- We can try to build better objects in code using object-oriented languages/techniques.

Imperative Programming Model

- Example: rotate a cube.
- The rotation function must know how the cube is represented.
  - Vertex list
  - Edge list

Object-Oriented Programming Model

- In this model, the representation is stored with the object.
- The application sends a message to the object.
- The object contains functions (methods) which allow it to transform itself.
C/C++

- Can try to use C structs to build objects
- C++ provides better support
  - Use class construct
  - Can hide implementation using public, private, and protected members in a class
  - Can also use friend designation to allow classes to access each other

Cube Object

- Suppose that we want to create a simple cube object that we can scale, orient, position and set its color directly through code such as
  ```cpp
cube mycube;
mycube.color[0]=1.0;
mycube.color[1]= mycube.color[2]=0.0;
mycube.matrix[0][0]=......
```

Cube Object Functions

- We would also like to have functions that act on the cube such as
  ```cpp
  mycube.translate(1.0, 0.0, 0.0);
  mycube.rotate(theta, 1.0, 0.0, 0.0);
  setcolor(mycube, 1.0, 0.0, 0.0);
  ```
- We also need a way of displaying the cube
  ```cpp
  mycube.render();
  ```

Building the Cube Object

```cpp
class cube {
  public:
    float color[3];
    float matrix[4][4];
    // public methods
    void translate(float x, float y, float z);
    void rotate(float theta, float x, float y, float z);
    void setcolor(float r, float g, float b);
    void render();
  }
  private:
  // implementation
}
```

The Implementation

- Can use any implementation in the private part such as a vertex list
- The private part has access to public members and the implementation of class methods can use any implementation without making it visible
- Render method is tricky but it will invoke the standard OpenGL drawing functions

Other Objects

- Other objects have geometric aspects
  - Cameras
  - Light sources
- But we should be able to have nongeometric objects too
  - Materials
  - Colors
  - Transformations (matrices)
Application Code

cube mycube;
material plastic;
mycube.setMaterial(plastic);

camera frontView;
frontView.position(x ,y, z);

Light Object

class light {
    // match Phong model
    public:
        boolean type; //ortho or perspective
        boolean near;
        float position[3];
        float orientation[3];
        float specular[3];
        float diffuse[3];
        float ambient[3];
}

Scene Descriptions

- If we recall figure model, we saw that
  - We could describe model either by tree or by equivalent code
  - We could write a generic traversal to display
- If we can represent all the elements of a scene (cameras, lights, materials, geometry) as C++ objects, we should be able to show them in a tree
  - Render scene by traversing this tree

Another Example

struct IndexedPrimitive {
    point3 *pts;
    point3 *normals;
    point2 *tCoords;
    int *indices;
    int NPts;
    GLuint VBO;
    int NIndices;
    GLuint IBO;
    void SetupIndexedBuffers();
    void Bind(GLuint program);
};

Another Example

class Torus: public IndexedPrimitive {
    public:
        Torus() {
            GeneratePts(10, 10);
            SetupIndexedBuffers();
        }
        Torus(int s) {
            GeneratePts(s, s);
            SetupIndexedBuffers();
        }
        Torus(int i, int j) {
            GeneratePts(i, j);
            SetupIndexedBuffers();
        }
        void GeneratePts(int, int);
};
Another Example

class Cube: public Primitive {
   void GenerateCube(Primitive *);

   public:
   Cube() {
      GenerateCube(this);
      SetupPrimitiveBuffers();
   }
}

Another Example

class Plane: public IndexedPrimitive {
   void render(Transform xfm, Matrix projection, GLuint program);
   ...
}

Another Example

Another Example

Materials

void setMaterial(Material *mat, float dr, float dg, float db, float da, float sr, float sg, float sb, float sa, float p) {
   setclr(mat->diffuse, dr, dg, db, da);
   setclr(mat->specular, sr, sg, sb, sa);
   mat->shiny = p;
}

void setMaterial2(Material mat, color4 d, color4 s, float p) {
   clrcpy(mat.diffuse, d);
   clrcpy(mat.specular, s);
   mat.shiny = p;
}

Another Vertex Shader

// Put normal into world space.
ambient = NormalMatrix * NormalMatrix * NormalMatrix * NormalMatrix;
// Put normal into eye space.
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
ambient = NormalMatrix * Ambient;
ambient = LightPosition - ModelView * vNormal;
ambient = ModelView * vNormal;
#else

// In clip space!
TC = vTex;

gl_Position = Projection*ModelView*vPosition;
TN = (Projection*NormalMatrix*vec4(vNormal,0)).xyz;

FE = gl_Position.xyz;
FL = (Projection*LightPosition).xyz;

if (LightPosition.w != 0.0) {
    FL = (Projection*LightPosition).xyz - FE;
}
#endif

---

Scene Graph

![Scene Graph Diagram]

Preorder Traversal

PushAttrib
PushMatrix
Color
Translate
Rotate
Object1
Translate
Object2
PopMatrix
PopAttrib

Group Nodes

- Necessary to isolate state changes
- Equivalent to Push/Pop
- Note that as with the figure model
  - We can write a universal traversal algorithm
  - The order of traversal can matter

- If we do not use the group node, state changes can persist

---

Scene Graph Possibilities

- Open Inventor – Mostly dead, but TalkingHead uses it 😊
- Some alternatives that are active
  - Coin3D
  - VSG (commercial)
- OpenSceneGraph – Still Active
- Java3D
- Web Based
  - X3D – Successor to VRML
  - O3D – (Google) JavaScript API
  - 3DMLW
- Collada – XML scheme from Khronos Group
- RenderMan
- Acrobat 3D
- OpenSG – based on OpenGL – not sure alive