CS354 Computer Graphics
Character Animation and Skinning

Qixing Huang
April 9th 2018
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
Structure Through Function Calls

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

Chassis

- Right-front wheel
- Left-front wheel
- Right-rear wheel
- Left-rear wheel
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: $\mathbf{R}_b$
  - Apply $\mathbf{M} = \mathbf{R}_b$ to base
- Translate lower arm relative to base: $\mathbf{T}_{lu}$
- Rotate lower arm around joint: $\mathbf{R}_{lu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu}$ to lower arm
- Translate upper arm relative to upper arm: $\mathbf{T}_{uu}$
- Rotate upper arm around joint: $\mathbf{R}_{uu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu} \mathbf{T}_{uu} \mathbf{R}_{uu}$ to upper arm
robot_arm()
{
    glRotate(theta, 0.0, 1.0, 0.0);
    base();
    glTranslate(0.0, h1, 0.0);
    glRotate(phi, 0.0, 0.0, 1.0);
    lower_arm();
    glTranslate(0.0, h2, 0.0);
    glRotate(psi, 0.0, 0.0, 1.0);
    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

- **Draw**: Code for drawing part or pointer to drawing function
- **M**: Linked list of pointers to children
- **Child**: 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
Tree with Matrices
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
  - $M_{lua}$, $M_{rua}$, $M_{lul}$, $M_{rul}$ position arms and legs 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 $M M_h$ and draw head
- For left-upper arm need $M M_{lua}$ and so on
- Rather than recomputing $M M_{lua}$ from scratch or using an inverse matrix, we can use the matrix stack to store $M$ and other matrices as we traverse the tree
Traversal Code

```c
figure() {
  glPushMatrix();
  torso();
  glRotate3f(...);
  head();
  glPopMatrix();
  glPushMatrix();
  glTranslate3f(...);
  glRotate3f(...);
  left_upper_arm();
  glPopMatrix();
  glPushMatrix();
```

- save present model-view matrix
- update model-view matrix for head
- recover original model-view matrix
- save it again
- update model-view matrix for left upper arm
- recover and save original model-view matrix again
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
Skinning and Character Animation
Objectives

• Introduce the basics of character animation
• Introduce skinning
• Introduce basic linear blend skinning
Character Animation

• Skeletons and skin
  – skeleton – a hierarchy of bones or joints
  – note arrows pointing from parent to child joint
  – skin – the polygon mesh defining the body surface
Binding

• Define transform between joint and skin spaces in rest or bind pose

• Associate skin vertices to subset of the joints
Animation

• Move the joints and the skin moves with them

• This deforms the mesh from its rest position
Skin

- Skin is a set of polygonal meshes
- A mesh is a collection of (connected) polygons
Skin

- A skin mesh is defined in its own local frame.
Binding

- Each joint (bone) has its own local frame
- Let $B_j$ be the transformation from local joint frame $j$ to the skin mesh local frame in the binding pose
- $B_j$ is represented by a binding matrix

![Diagram](image.png)
Rigid skinning – basic idea

- Associate a group of vertices to a single joint $j$
- Let $T_j$ be the transformation from joint $j$ local space to world space
- Then the skin vertex transform to world space for vertices $v_k$ associated with joint $j$ is $v_k^T = T_j B_j^{-1} v_k$
Joint motion

- When joint $j$ moves, $T_j$ changes and the skin vertices move with it.
- The relative positions of the vertices in the local joint frame don’t change.
- $v_k^T = T_j B_j^{-1} v_k$
Problems with rigid skinning

- Simple but low quality because large distortions happen when bends form at joints

Parag Chaudhuri, 2012
Linear Blending Skinning

• Adds flexibility to fix artifacts but still simple and fast
• Commonly used in games
• Vertices associated with multiple joints, not just one
• Vertex transform is a linear combination of the transforms associated with its joints. Each vertex has weights for this linear combination assigned to it

\[ v'_k = \sum_i w_{i,k} T_i B_i^{-1} v_k \]

\[ \forall k \sum_i w_{i,k} = 1 \text{ and } 0 \leq w_{i,k} \leq 1 \]

• Vertex normal can be computed similarly

Parag Chaudhuri, 2012
Fewer artifacts

• With proper weights many but not all artifacts are eliminated or improved
Linear blend skinning algorithm

• Skin::Update()
  – Compute $M_i = T_i B_i^{-1}$ for each joint. Note that $B_i^{-1}$ can be precomputed and stored. For each vertex compute world position and normal.

• Skin::Draw()
  – Initialize ModelView matrix.
  – Draw skin polygons using global positions and vertices.
Problems

- Skin collapse at bends

Figure 1: The skeleton subspace deformation algorithm. The deformed position of a point $p$ lies on the line $p'p''$ defined by the images of that point rigidly transformed by the neighboring skeletal coordinate frames, resulting in the characteristic “collapsing elbow” problem (solid line).

Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation, Lewis, Cordner and Fong, SIGGRAPH 2000
Problems

- Skin collapses at twists

Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation, Lewis, Cordner and Fong, SIGGRAPH 2000
Dual Quaternion Skinning

• Better solution, nearly as fast

Geometric Skinning with Approximate Dual Quaternion Blending, Kavan Collins, Zara and O'Sullivan, ACMTOG 2008
Linear Blend Skinning

• Problems
  – Binding is difficult – what joints should each vertex be associated with?
  – Weight assignment is not intuitive and very time-consuming
  – Still have collapse with linear blend skinning

• Advantages
  – Simple
  – Fast
  – Easy GPU implementation