Computer Animation

Michael Kazhdan

(601.457/657)

HB 16.5, 16.6
FvDFH 21.1, 21.3, 21.4
Announcements

• The assignment 3 webpage has links to:
  ◦ Slides describing OpenGL
  ◦ A video tutorial on OpenGL
Overview

• Some early animation history
  ◦ http://web.inter.nl.net/users/anima/index.htm
  ◦ http://www.public.iastate.edu/~rllew/chrnearl.html

• Computer animation

• Assignment 3
Thaumatrope

• Why does animation work?
• Persistence of vision
• 1824 John Ayerton invents the *thaumatrope*
• Or, 1828 Paul Roget invents the *thaumatrope*
Phenakistoscope

- Invented independently by 2 people in 1832
- Disc mounted on spindle
- Viewed through slots with images facing mirror
- Turning disc animates images
Phenakistoscope

- Invented independently by 2 people in 1832
- Disc mounted on spindle
- Viewed through slots with images facing mirror
- Turning disc animates images
Phenakistoscope

- Invented independently by 2 people in 1832
- Disc mounted on spindle
- Viewed through slots with images facing mirror
- Turning disc animates images
Zoetrope (1834)

- Images arranged on paper band inside a drum
- Slits cut in the upper half of the drum
- Opposite side viewed as drum rapidly spun
- Praxinoscope is a variation on this
Zoetrope (1834)

- Images arranged on paper band inside a drum
- Slits cut in the upper half of the drum
- Opposite side viewed as drum rapidly spun
- Praxinoscope is a variation on this
Mutoscope (1895)

- Coin-operated “flip-book” animation
- Picture cards attached to a drum
- Popular at sea-side resorts, etc.
Animation History

First known example of animation:

• “Humorous Phases of Funny Faces” (1906)
Key Developments

- Plot
- Creation of animation studios
- Inking on cels

“Felix the Cat”  
Otto Messmer (1921)

“Steamboat Willie”  
Walt Disney (1928)

“Gertie the Dinosaur”  
Windsor McCay (1914)
Key Developments

• Max Fleischer invents rotoscoping (1921)
Key Developments

• Max Fleischer invents rotoscoping (1921)
Key Developments

• “Flowers and Trees”, 1932:
  ◦ Uses color, wins Academy Award

• “Snow White” (aka “Disney’s Folly”), 1937:
  ◦ $1.4 million to make
  ◦ 750 artists
  ◦ Highest grossing ($8 million)
Animation Uses

- Entertainment
- Education
- Propaganda
Principles of Traditional Animation

How do we communicate aspects of the animation that are not strictly visual?

- Rigidity
- Weight
- Mood
- Intent
- Focus
- Etc.

See, for example, *The Illusion of Life: Disney Animation* for Disney’s 12 basic principals of animation.

Luxo Junior

https://en.wikipedia.org/wiki/12_basic_principals_of_animation
Overview

• Some early animation history

• Computer animation
  ○ Keyframe animation
  ○ Articulated figures
  ○ Kinematics and dynamics

• Assignment 3
Keyframe Animation

• Define character poses at specific time steps called “keyframes”
Keyframe Animation

• Interpolate variables describing keyframes to determine poses for character “in-between”
Articulated Figures

- Character poses described by set of rigid bodies connected by “joints”

Scene Graph

Angel Figures 8.8 & 8.9
Articulated Figures

- Well-suited for humanoid characters

```
Root
  ↓
Chest
  ↓
Neck  LCollar  RCollar
  ↓  ↓  ↓
Head  LShld  RShld
  ↓  ↓  ↓
LElbow  RElbow
  ↓  ↓
LWrist  RWrist
  ↓
LHip  RHip
  ↓  ↓
LKnee  RKnee
  ↓  ↓
LAnkle  RAnkle
```

Rose et al. `96
Example: Walk Cycle

• Articulated figure:
Example: Walk Cycle

- Hip joint orientation:
Example: Walk Cycle

• Knee joint orientation:
Example: Walk Cycle

- Ankle joint orientation:
Example: Walk Cycle

http://www.ischool.utexas.edu/~luna73/architecture/
Keyframe Animation

• In-betweening (translation):
  ◦ Cardinal B-splines – maybe not be good enough
    » May not follow physical laws

Recall: Convex hull containment
Articulated Figures

- In-betweening (rotation)
  - If you interpolate vertex positions (e.g. instead of angles) the geometry may get distorted.

Good arm  Bad arm
Kinematics and Dynamics

• Kinematics: *Study of motion w/o regard for the cause*
  - Considers only motion
  - Determined by positions, velocities, accelerations

• Dynamics: *Study of the cause of motion*
  - Considers underlying forces and interactions
  - Compute motion from initial conditions and physics
Example: 2-Link Structure

- Two links connected by rotational joints

$$\Theta_1$$

$$\Theta_2$$

$$l_1$$

$$l_2$$

$$(0,0)$$

$$X = (x,y)$$

"End-Effector"
Forward Kinematics

- Animator specifies joint angles: $\Theta_1$ and $\Theta_2$
- Computer finds positions of end-effector: $X$

$$X = (l_1 \cos \Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin \Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$$
Forward Kinematics

- Joint motions can be specified by spline curves

\[ X = (x, y) \]
Example: 2-Link Structure

- What if animator knows position of “end-effector”
Inverse Kinematics

- Animator specifies end-effector positions: \( X \)
- Computer finds joint angles: \( \Theta_1 \) and \( \Theta_2 \):

\[
\Theta_2 = \cos^{-1}\left( \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)
\]

\[
\Theta_1 = \frac{-(l_2 \sin(\Theta_2)x + (l_1 + l_2 \cos(\Theta_2))y)}{(l_2 \sin(\Theta_2))y + (l_1 + l_2 \cos(\Theta_2))x}
\]

\( X = (x,y) \)
Inverse Kinematics

- End-effector positions can be specified by splines

\[ X = (x, y) \]
Inverse Kinematics

- Problem for more complex structures
  - System of equations is usually under-defined
  - Multiple (or no) solutions

Three unknowns: $\Theta_1, \Theta_2, \Theta_3$
Two equations: $x, y$
Inverse Kinematics

- Solution for more complex structures:
  - Find best solution (e.g., minimize energy in motion)
  - Non-linear optimization

\[ X = (x, y) \]
Summary of Kinematics

• Forward kinematics
  ◦ Specify conditions (joint angles)
  ◦ Compute positions of end-effectors

• Inverse kinematics
  ◦ “Goal-directed” motion
  ◦ Specify goal positions of end effectors
  ◦ Compute conditions required to achieve goals

Inverse kinematics provides easier specification for many animation tasks, but it is computationally more difficult
Dynamics

- Simulation of physics insures realism of motion
Spacetime Constraints

• Animator specifies constraints:
  ◦ What the character’s physical structure is
    » e.g., articulated figure
  ◦ What the character has to do
    » e.g., jump from here to there within time $t$
  ◦ What other physical structures are present
    » e.g., floor to push off and land
  ◦ How the motion should be performed
    » e.g., minimize energy
Spacetime Constraints

- Computer finds the “best” physical motion satisfying the constraints

- Example: particle with jet propulsion
  - $x(t)$ is position of particle at time $t$
  - $f(t)$ is the directional force of jet propulsion at time $t$
  - Particle’s equation of motion is:
    \[ 0 = m(x'' - g) - f \]
  - Move from $a$ to $b$ within $t_0$ to $t_1$, minimizing
    \[ \int_{t_0}^{t_1} |f(t)|^2 dt \]
  - Such that:
    \[ x(t_0) = a, \quad x'(t_0) = 0, \quad x(t_1) = b, \quad \text{and} \quad x'(t_1) = 0 \]
Spacetime Constraints

Discretize time steps \( \{x_0, \cdots, x_N\} \):

\[
x'_i = \frac{x_i - x_{i-1}}{h}
\]

\[
x''_i = \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2}
\]

\[
f_i = m \left( \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} - g \right)
\]

Minimize

\[
\int_{t_0}^{t_1} |f(t)|^2 dt \approx h \sum_i |f_i|^2 = hm^2 \sum_i \left\| \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} - g \right\|^2
\]

subject to \( x_{-1} = x_0 = a \) and \( x_N = x_{N+1} = b \).

Witkin & Kass '88
Spacetime Constraints

For simple scenarios:
- Solve a linear system
  \[ Ax = b \]

For complex scenarios:
- Solve using iterative optimization techniques

Witkin & Kass ‘88
Spacetime Constraints

• Advantages:
  ◦ Free animator from having to specify details of physically realistic motion with spline curves
  ◦ Easy to vary motions due to new parameters and/or new constraints

• Challenges:
  ◦ Specifying constraints and objective functions
  ◦ Avoiding local minima during optimization
Dynamics

• Other physical simulations:
  ○ Rigid bodies
  ○ Soft bodies
  ○ Cloth
  ○ Liquids
  ○ Gases
  ○ etc.

Hot Gases
(Foster & Metaxas `97)

Cloth
(Baraff & Witkin `98)
Overview

• Some early animation history

• Computer animation

• Assignment 3
  ◦ General Structure
  ◦ The Matrix Stack
  ◦ Drawing Primitives
Assignment 3 (General Structure)

Window::DisplayFunction:

At every frame, the code redraws the display window by invoking Window::DisplayFunction. This function:

1. Clears the buffers
2. Sets up the projection matrix
3. Invokes Scene::drawOpenGL

This is taken care of for you.
Assignment 3 (General Structure)

**Scene::drawOpenGL:**

When invoked, this function

1. Draws the camera (**Camera::drawOpenGL**)
2. Draws the lights (**Light::drawOpenGL**)
3. Draws the geometry (**SceneGeometry::drawOpenGL**)
   » Which draws the shapes (**ShapeList::drawOpenGL**)

The calls to these functions are made for you, but you have to implement them.
Assignment 3 (Matrix Stack)

In OpenGL, the model-to-camera transformation are stored in the `GL_MODELVIEW` matrix stack.

The matrix at the top of the stack describes the transformation that will be applied to the subsequently drawn geometry (vertex positions and normal) to bring it into the camera’s coordinate system.
Assignment 3 (Matrix Stack)

In OpenGL, the model-to-camera transformation are stored in the *GL_MODELVIEW* matrix stack.

You can push/pop matrices onto the stack:

- `glPushMatrix`: Pushes the matrix at the top of the stack onto the matrix stack (increasing the stack size by one)
- `glPopMatrix`: Pops the top matrix off of the stack
In OpenGL, the model-to-camera transformation are stored in the `GL_MODELVIEW` matrix stack.

You can set the matrix at the top of the stack by:

- `glLoadMatrix*`: sets it to the prescribed matrix
- `glLoadIdentity`: sets it to the identity matrix
Assignment 3 (Matrix Stack)

In OpenGL, the model-to-camera transformation are stored in the GL_MODELVIEW matrix stack.

You can multiply the matrix at the top of the stack on the right by:

- `glMatrixMode*`: multiplies by the prescribed matrix
- `glTranslatef*/glRotate*`: multiplies by the matrix describing the translation/rotation
- `gluLookAt*`: multiplies by the matrix describing a camera with the prescribed orientation
Scene Graphs (Recall):

In a scene graph, edges are tagged with a local-to-global transformations.

The transformation taking a node in a scene graph into global coordinates is the composition of the transformations from the root to the node.
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

To transform Desk1 into the Building’s coordinate system we:

- Transform it into Office1’s coordinates ($M_8$), then
- Transform it into Floor2’s coordinates ($M_6$), then
- Transform it into Building’s coordinates ($M_2$)

This gives:

$$M_2 \circ M_6 \circ M_8$$
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

To transform Desk1 into the Building’s coordinate system we:

- Transform it into Office1’s coordinates \((M_9)\), then
- Transform it into Floor2’s coordinates \((M_6)\), then
- Transform it into Building’s coordinates \((M_2)\)

This gives:

\[ M_2 \circ M_6 \circ M_9 \]
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

When drawing Office1, we have $M_2 \circ M_6$ at the top of the stack.

Before drawing Bookshelf1, we want $M_2 \circ M_6 \circ M_8$ at the top of the stack.

Before drawing Desk1, we want $M_2 \circ M_6 \circ M_9$ at the top of the stack.
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
- Draw *Bookshelf1*
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
- Draw Bookshelf1
- Multiply on the right by $M_8^{-1}$
Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
- Draw Bookshelf1
- Multiply on the right by $M_8^{-1}$
- Multiply on the right by $M_9$
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
- Draw Bookshelf1
- Multiply on the right by $M_8^{-1}$
- Multiply on the right by $M_9$
- Draw Desk1

Matrix stack:

$M_2 \circ M_6 \circ M_8 \circ M_8^{-1} \circ M_9 \circ \cdots$
Scene Graphs (Recall):

We could:

- Multiply on the right by $M_8$
- Draw Bookshelf1
- Multiply on the right by $M_8^{-1}$
- Multiply on the right by $M_9$
- Draw Desk1
- Multiply on the right by $M_9^{-1}$
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could:
- Multiply on the right by $M_8$
- Draw Bookshelf1
- Multiply on the right by $M_8^{-1}$
- Multiply on the right by $M_9$
- Draw Desk1
- Multiply on the right by $M_9^{-1}$
- Etc.

The accumulation of matrix products, combined with numerical imprecision, could produce the wrong results.
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
- Pop off the top of the stack
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
- Pop off the top of the stack
- Push \((M_2 \circ M_6)\) onto the stack
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \( (M_2 \circ M_6) \) onto the stack
- Multiply on the right by \( M_8 \)
- Draw Bookshelf1
- Pop off the top of the stack
- Push \( (M_2 \circ M_6) \) onto the stack
- Multiply on the right by \( M_9 \)
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
- Pop off the top of the stack
- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_9\)
- Draw Desk1
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
- Pop off the top of the stack
- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_9\)
- Draw Desk1
- Pop off the top of the stack
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

We could also:

- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_8\)
- Draw Bookshelf1
- Pop off the top of the stack
- Push \((M_2 \circ M_6)\) onto the stack
- Multiply on the right by \(M_9\)
- Draw Desk1
- Pop off the top of the stack
- Etc.
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

While all of this can be used to tell OpenGL how to transform the geometry from local coordinates to global coordinates, you actually want to tell OpenGL how to transform the geometry from local coordinates to camera coordinates.
Assignment 3 (Matrix Stack)

Scene Graphs (Recall):

To do this, you want to multiply the local-to-global transformation on the left by the global-to-camera transformation.

⇔ Make the global-to-camera transformation the first thing on the stack.
   - Make sure to draw your camera first.
   - Make sure you are setting the matrix stack with the global-to-camera transformation, not multiplying by it.
   - Make sure you are working with the modelview matrix stack.
Assignment 3 (Drawing Primitives)

**glVertex**:  
When you invoke this function, OpenGL sends the vertex into the rendering pipeline with:  
- Position obtained by applying the current modelview transform  
- Color computed using the current lights and materials  
- Texture coordinates as specified

**OpenGL is a state machine**:  
OpenGL will always use the last specified normals, lights, material properties, etc. even if you did not specify them explicitly.
**glVertex**: 
When you invoke this function, OpenGL sends the vertex into the rendering pipeline with:

- Position obtained by applying the current modelview transform
- Color computed using the current lights and materials
- Texture coordinates as specified

**OpenGL is a state machine:**
OpenGL will always use the last specified normals, lights, material properties, etc. even if you did not specify them explicitly.

Make sure to set the vertex’s properties before specifying its position!