Lecture 12

2D Animation
Animation Basics: The FilmStrip

- Animation is a sequence of **hand-drawn frames**
  - Smoothly displays action when change quickly
  - Also called flipbook animation

- Arrange animation in a **sprite sheet** (one texture)
  - Software chooses which frame to use at any time
  - So programmer is actually the one doing animation
Anatomy of FilmStrip Class

/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 */

void FilmStrip::setFrame(int frame) {
    this->frame = frame;
    int x = (frame % cols)*rect.size.width;
    int y = (frame / cols)*rect.size.height;
    rect.origin.set(x,y);
    setTextureRect(rect);
}

Scene Graphs
Anatomy of FilmStrip Class

/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 *
 */

void FilmStrip::setFrame(int frame) {
    this->frame = frame;
    int x = (frame % cols)*rect.size.width;
    int y = (frame / cols)*rect.size.height;
    rect.origin.set(x, y);
    setTextureRect(rect);
}

Method in Sprite to set the part of texture drawn.
/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 *
 */

void FilmStrip::setFrame(
    int frame) {
    this->frame = frame;
    int x = (frame % cols)*rect.size.width;
    int y = (frame / cols) * rect.size.height;
    rect.origin.set(x,y);
    setTextureRect(rect);

Adjusting your Speed

- Do not want to go too fast
  - 1 animation frame = 16 ms
  - Walk cycle = 8/12 frames
  - Completed in 133-200 ms

- General solution: cooldowns
  - Add an int timer to your object
  - Go to next frame when it is 0
  - Reset it to > 0 at new frame

- Simple but tedious
  - Have to do for each object
  - Assumes animation is in a loop
Combining Animations

- Characters to a lot of things
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while player AFK

- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?

- **Idea**: shared boundaries
  - End of loop = start of another
  - Treat like advancing a frame
Combining Animations

- Characters to a lot of things
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while player AFK

- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?

**Idea**: shared boundaries
  - End of loop = start of another
  - Treat like advancing a frame
Combining Animations

- Characters to a lot of things
  - Run, jump, duck, slide
  - Fire weapons, cast spells
  - Fidget while player AFK
- Want animations for all
  - Is loop appropriate for each?
  - How do we transition?
- **Idea:** shared boundaries
  - End of loop = start of another
  - Treat like advancing a frame
Animation and State Machines

- Idea: Each sequence a state
  - Do sequence while in state
  - Transition when at end
  - Only loop if loop in graph

- A graph edge means…
  - Boundaries match up
  - Transition is allowable

- Similar to data driven AI
  - Created by the designer
  - Implemented by programmer
  - Modern engines have tools
Animation and State Machines

- Idea: Each sequence a state
  - Do sequence while in state
  - Transition when at end
  - Only loop if loop in graph
- A graph edge means…
  - Boundaries match up
  - Transition is allowable
- Similar to data driven AI
  - Created by the designer
  - Implemented by programmer
  - Modern engines have tools

Scene Graphs
Complex Example: Jumping

- **stand**
  - **stand2crouch**
    - **crouch**
    - **takeoff**
    - **hop**
    - **float**
  - **land**
Complex Example: Jumping

- Stand
- Stand2Crouch
- Crouch
- Takeoff
- Near Ground
- Hop
- Float
- Land
- Jump Press
- Jump Release
Complex Example: Jumping

Transition state needed to align the sequences

Stand

Stand2crouch

Crouch

Takeoff

Hop

Float

Land
Aside: Sync Kills
The Responsiveness Issue

Tightness of the gameplay

Additional delay preventing jump

Scene Graphs
Fast Transitions: Crossfade Blending

- Linear interpolation on colors

\[
\begin{align*}
    r_c &= tr_a + (1 - t)r_b \\
    g_c &= tg_a + (1 - t)g_b \\
    b_c &= tb_a + (1 - t)b_b
\end{align*}
\]

Note weights sum to 1.0
Fast Transitions: Crossfade Blending

- Linear interpolation on colors

\[
\begin{align*}
    r_c &= tr_a + (1 - t)r_b \\
    g_c &= tg_a + (1 - t)g_b \\
    b_c &= tb_a + (1 - t)b_b
\end{align*}
\]

Note weights sum to 1.0
Fast Transitions: Crossfade Blending

- Linear interpolation on colors
  
  \[ r_c = tr_a + (1 - t)r_b \]
  
  \[ g_c = tg_a + (1 - t)g_b \]
  
  \[ b_c = tb_a + (1 - t)b_b \]

  Note weights sum to 1.0
Fast Transitions: Crossfade Blending

- Linear interpolation on colors

\[
\begin{align*}
    r_c &= tr_a + (1 - t)r_b \\
    g_c &= tg_a + (1 - t)g_b \\
    b_c &= tb_a + (1 - t)b_b \\
\end{align*}
\]

Note weights sum to 1.0
Fast Transitions: Crossfade Blending

- Linear interpolation on colors

\[
\begin{align*}
    r_c &= tr_a + (1 - t)r_b \\
    g_c &= tg_a + (1 - t)g_b \\
    b_c &= tb_a + (1 - t)b_b
\end{align*}
\]

Note weights sum to 1.0
Combining With Animation

Cycle the filmstrip normally

Cycle the filmstrip normally

Combine with alpha blending
Related Concept: **Tweening**

- Act of linear interpolating between animation frames
  - Because we cycle filmstrip slower than framerate
  - Implements a form of motion blur

- If animation **designed right**, makes it smoother
Tweening Works for Transforms Too

- Any transform is represented by a **matrix**
  - Can linearly interpolate matrix components
  - Gives a reasonable transform “in-between”

- **Aside**: This is a motivation for **quaternions**
  - Gives smoother interpolation for rotation
Cocos2D Engine Support

- **Recall**: decoupled renderer
  - **Update** pass modifies scene
  - **Render** pass to SpriteBatch
  - **Draw** pass sends to GPU

- Render pass can animate
  - Switches frames if needed
  - Tweens in-between frames

- Purpose of **action API**
  - Give action with duration
  - Can be new frame
  - Also could be a transform
Executing Actions: Transforms

Action* action;
action = RotateBy::create(2.0f, 90.0f);
action->retain(); // Need it for later
sprite->runAction(action);

...

if (action->isDone()) {
    action->release();
    action = nullptr;
    // Do other clean-up
}

Scene Graphs
Executing Actions: Transforms

Action* action;
action = RotateBy::create(2.0f, 90.0f);
action->retain(); // Need it for later
sprite->runAction(action);

if (action->isDone()) {
    action->release();
    action = nullptr;
    // Do other clean-up
}

How long to spend

Tweens rotation
Executing Actions: FilmStrips

Action* action;

Vector<SpriteFrame*>* theFrames
auto f1 = SpriteFrame::create(im, rec1)
theFrames->push_back(f1);
...
auto f8 = SpriteFrame::create(im, rec8)
theFrames->push_back(f8);

Animation* seq = Animation::
    createWithSpriteFrames(theFrames, 0.1f);
action = Animate::create(seq);
action->retain();

sprite->runAction(action)

// Clean up same as before
Executing Actions: **FilmStrips**

```cpp
Action* action;

Vector<SpriteFrame*>* theFrames;
auto f1 = SpriteFrame::create(im, rec1);
theFrames->push_back(f1);
...
auto f8 = SpriteFrame::create(im, rec8);
theFrames->push_back(f8);

Animation* seq = Animation::
cREATEWITHSPRITEFRAMES(theFrames, 0.1f);
action = Animate::create(seq);
action->retain();
sprite->runAction(action);
```

// Clean up same as before

**Does not tween**

**Seconds per frame**
Executing Actions: FilmStrips

Action* action;

Vector<SpriteFrame*>* theFrames;
auto f1 = SpriteFrame::create(im, rec1);
theFrames->push_back(f1);
...
auto f8 = SpriteFrame::create(im, rec8);
theFrames->push_back(f8);

Animation* seq = Animation::
    createWithSpriteFrames(theFrames, 0.1f);
action = Animate::create(seq);
action->retain();
sprite->runAction(action);

// Clean up same as before

This is why I made FilmStrip
- Basic approach to tweening
  - Specify duration to animate
  - Set \( t = 0 \) at beginning
  - Normalize \( t = 1 \) at end
  - Interpolate value with \( t \)

- How does \( t \) change?
  - Usually done \textit{linearly}
  - Could be some other way

- \textbf{Easing}: how to change \( t \)
  - Used for bouncing effects
  - Best used for \textit{transforms}
Easing Function

- Basic approach to tweening
  - Specify duration to animate
  - Set $t = 0$ at beginning
  - Normalize $t = 1$ at end
  - Interpolate value with $t$

- How does $t$ change?
  - Usually done *linearly*
  - Could be some other way

- **Easing**: how to change $t$
  - Used for bouncing effects
  - Best used for *transforms*
Easing Functions in Cocos2D
Problem With Decoupled Animation

```cpp
Action* action;
action = RotateBy::create(2.0f, 90.0f);
action->retain(); // Need it for later
sprite->runAction(action);
```

What if we change our mind before 2 seconds?
Problem With Decoupled Animation

Action* action;
action = RotateBy::create(2.0f, 90.0f);
action->retain();  // Need it for later
sprite->runAction(action);

Compatible: Combine
Incompatible: Replace
Action* action;
action = RotateBy::create(2.0f, 90.0f);
action->retain();  // Need it for later
sprite->runAction(action);

Compatible: Combine
Incompatible: Replace

But multiple actions can make clean-up a mess
Final Word on Cocos2D Tweening

Transform Tweening

+  

Physical Animation

=  

Complete Disaster
Recall: Modular Animation

- Break asset into parts
  - Natural for joints/bodies
  - Animate each separately
- Cuts down on filmstrips
  - Most steps are transforms
  - Very natural for tweening
  - Also better for physics
- Several tools to help you
  - Example: *Spriter*
  - Great for visualizing design
Recall: Modular Animation

- Break asset into parts
  - Natural for joints/bodies
  - Animate each separately
- Cuts down on filmstrips
  - Most steps are transforms
  - Very natural for tweening
  - Also better for physics
- Several tools to help you
  - Example: Spriter
  - Great for visualizing design
Recall: Modular Animation

- Break asset into parts
  - Natural for joints/bodies
  - Animate each separately
- Cuts down on filmstrips
  - Most steps are transforms
  - Very natural for tweening
  - Also better for physics
- Several tools to help you
  - Example: *Spriter*
  - Great for visualizing design
Recall: Modular Animation

- Break asset into parts
  - Natural for joints/bodies
  - Animate each separately
- Cuts down on filmstrips
  - Most steps are transforms
  - Very natural for tweening
  - Also better for physics
- Several tools to help you
  - Example: Spriter
  - Great for visualizing design

- Inside hit box can safely
  - Transform with duration
  - Tween animations
  - Manage multiple actions
Aside: Skinning
Aside: Skinning

Way to get extra usage of hand-drawn frames
Spriter Demo
Basic Idea: Bones
Basic Idea: Bones
Basic Idea: Bones

- Pivot (origin)
- Orientation ($y$-axis)
- Sprite attached

Creates implicit coordinate space
Bones are Hierarchical
Bones are Heirarchical

Transforms apply to children
Bones are Hierarchical

Transforms do not affect the parent
Recall: Scene Graph Hierarchy

Layer

Node

Node

Node

Node

Node

Node

Device/Screen Coordinates

Bounded box inside

Coords relative to parent box
Bones are a Scene Graph Visualization
Manage With Multiple State Machines

- legs idle
  - legs walk
  - arms idle
    - arms shoot
Manage With Multiple State Machines

Can be independent or coordinated

- legs idle
- legs walk
- arms idle
- arms shoot

Scene Graphs
Summary

- Standard 2D animation is **flipbook** style
  - Create a sequence of frames in sprite sheet
  - Switch between sequences with state machines

- **Tweening** supports interpolated transitions
  - Helpful for motion blur, state transitions
  - Transforms can be combined with easing functions

- Professional 2D animation uses **modular sprites**
  - Scene graphs are a simplified form of model rigging
  - State machine coordination can be very advanced