Lecture 11

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

void AnimationNode::setFrame(int frame) {
    this->frame = frame;
    int x = (frame % cols)*bounds.size.width;
    int y = (frame / cols)*bounds.size.height;
    bounds.origin.set(x,y);
    setPolygon(bounds);
}
/**
 * Sets the active frame as the given index.
 *
 * @param frame the index to make the active frame
 */

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

Landing Animation

Idling Animation
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
Complex Example: Jumping

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

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

Transition state needed to align the sequences
Aside: Sync Kills
The Responsiveness Issue

**Tightness of the gameplay**

- **stand**
- **stand2crouch**
- **crouch**
- **hop**
- **takeoff**
- **float**
- **land**

Additional delay preventing jump
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

\[t = 0.3\]
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

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

Note weights sum to 1.0

\( t = 1.0 \)
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
Supporting Tweened Animations

**Actions**
- Represents animation type
  - Moving, rotating, scaling
  - Filmstrip sequences
- But not active animation
  - Can be reused and replayed
  - Can be copied safely
- Think of as a “template”
  - Defines the tweening
  - But has no internal state

**ActionManager**
- Manages active animations
- Maps actions to scene graph
- Allocates animation state
- Has a separate update loop
  - Initialization step at start
  - Update step to increment
- Similar to asset manager
- Animations have key id
- Run update() to fit budget
Supporting Tweened Animations

**ActionManager**

- Manages active animations
- Maps actions to scene graph
- Allocates animation state

- Has a separate update loop
  - Initialization step at start
  - Update step to increment

- Similar to **asset manager**
  - Animations have key id
  - Run `update()` to fit budget
auto mgr = ActionManager::alloc();

auto action = RotateBy::alloc(90.0f, 2.0f);

mgr->activate(key, action, sprite);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

// No clean-up. Done automatically
Executing Actions: Transforms

auto mgr = ActionManager::alloc();

auto action = RotateBy::alloc(90.0f, 2.0f);

mgr->activate(key, action, sprite);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

// No clean-up. Done automatically

2D Animation
Executing Actions: **Transforms**

```cpp
auto mgr = ActionManager::alloc();
auto action = RotateBy::alloc(90.0f, 2.0f);
mgr->activate(key, action, sprite);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

// No clean-up. Done automatically
```

- How long to spend
- Tweens rotation
- Maps to framerate
Executing Actions: FilmStrips

```cpp
template{
    auto mgr = ActionManager::alloc();
    std::vector<int> frames;
    frames.push_back(f1);
    ...
    frames.push_back(f8);
    auto action = Animate::alloc(frames, 2.0f);
    mgr->activate(key, action, sprite);
    while (mgr->isActive(key)) {
        mgr->update(TIMESTEP);
    }
    // No clean-up. Done automatically
```
auto mgr = ActionManager::alloc();
std::vector<int> frames;
frames.push_back(f1);
...
frames.push_back(f8);
auto action = Animate::alloc(frames, 2.0f);
mgr->activate(key, action, sprite);
while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

// No clean-up. Done automatically

Executing Actions: FilmStrips

Sequence indices

Does not tween

Frames displayed uniformly
```cpp
auto mgr = ActionManager::alloc();
std::vector<int> frames;
frames.push_back(f1);
...
frames.push_back(f8);

auto action = Animate::alloc(frames, 2.0f);
mgr->activate(key, action, sprite);

while (mgr->isActive(key)) {
    mgr->update(TIMESTEP);
}

// No clean-up. Done automatically
```

Alternatively, could specify time per frame.
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*
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
Classic Easing Functions
Classic Easing Functions

http://easings.net
Problem With Decoupled Animation

```cpp
auto mgr = ActionManager::alloc();
auto action = RotateBy::alloc(90.0f, 2.0f);
mgr->activate(key, action, sprite);
```

What if we change our mind before 2 seconds?
auto mgr = ActionManager::alloc();
auto action = RotateBy::alloc(90.0f,2.0f);
mgr->activate(key,action,sprite);

Compatible: Combine
Incompatible: Replace
Problems With Decoupled Animation

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, Spine
  - 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, Spine
  - 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, Spine
  - 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

Loose hit boxes

- 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
Basic Idea: Bones
Basic Idea: Bones

2D Animation
Basic Idea: Bones

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

Creates implicit coordinate space
Bones are Hierarchical

Parent

Child
Bones are Heirarchical

Transforms apply to children
Bones are Hierarchical

Transforms do not affect the parent
Recall: Scene Graph Hierarchy

Bounded box inside

Device/Screen Coordinates

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

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

2D Animation
Manage With Multiple State Machines

Can be independent or coordinated

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

2D Animation
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