Lecture 9

Scene Graphs
Structure of a Cocos2D-x Application

Deployed Platform

App Delegate

Director

Scene

Scene

Scene
Structure of a Cocos2D-x Application

- Deployed Platform (iOS or Android)
- App Delegate
- Director
  - Scene
  - Scene
  - Scene

Memory policy (*future lecture*)
Structure of a Cocos2D-x Application

Deployed Platform

App Delegate

Access as a singleton

Director::getInstance()

Director

Scene

Scene

Scene

Scene Graphs
Structure of a Cocos2D-x Application

Deployed Platform

App Delegate

Director

Active

Dormant

Scene

Scene

Scene

Player Modes/Levels
The Cocos2D-x Director

• Sort of like a controller
  • Manages main game loop
  • Effectively owns everything

• It is a black-box **singleton**
  • You change settings…
  • but cannot add methods

• Adjustable settings
  • Current active scene
  • OpenGL drawing context
  • Cache of loaded textures
  • Input event listeners

Diagram:
- Director
- Scene
  - Custom code must go here

Scene Graphs
Aside: When Do We Load Assets?

Choice affects **design** & **ownership** of the asset manager

Deployment Platform

App Delegate

Application Start-up

Director

Level Load

Scene

Scene

Scene

Scene Graphs
The Scene Graph

Scene

Layer

Necessary to support touch input

Or any subclass

Node

Node

Node

Node

Node

Node

Node
Each Node is a Coordinate System

- Bounded box inside
- Device/Screen Coordinates
- Coords relative to parent box
Each Node is a Coordinate System
Each Node is a Coordinate System

Layer

Node

Node

Node

Layer

Node

Node

Node

Node

Node

Scene Graphs
Each Node is a Coordinate System
Settings Pass Down the Graph

Layer

Node

Node

Node

Transforms on parent also transform children
Settings Pass Down the Graph

Layer

Node

Node

Node

Transparency on parent also applies to children
Settings Pass Down the Graph

Disabling the parent also disables children
The Scene Graph

Necessary to support touch input
Creating a Custom Scene

- Engine has RootLayer class
  - Subclass of class Layer
  - Simplifies later subclassing

- Do not call its constructor!
  - Use function createScene
  - Call templated with class
  - **Example** (in AppDelegate):
    s = createScene<ShipRoot>()
  - In namespace GameRoot

- See the code samples
  - All follow this pattern

```cpp
class GameMode: public RootLayer {
private:
    // Internal controllers/models ...

public:
    /** Starts the game */
    void start() override;

    /** Update one frame */
    void update(float dt) override;

    /** Stops the game */
    void stop() override;
};
```
# The Two Main RootLayer Methods

<table>
<thead>
<tr>
<th><strong>start()</strong></th>
<th><strong>update()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Handles the game assets</td>
<td></td>
</tr>
<tr>
<td>• Attaches the asset loaders</td>
<td></td>
</tr>
<tr>
<td>• Loads immediate assets</td>
<td></td>
</tr>
<tr>
<td>• Starts any global singletons</td>
<td></td>
</tr>
<tr>
<td>• <strong>Example</strong>: SoundController</td>
<td></td>
</tr>
<tr>
<td>• Creates any player modes</td>
<td></td>
</tr>
<tr>
<td>• But does not launch <em>yet</em></td>
<td></td>
</tr>
<tr>
<td>• Waits for assets to load</td>
<td></td>
</tr>
<tr>
<td>• Like <code>GDXRoot</code> in 3152</td>
<td></td>
</tr>
</tbody>
</table>

• Called each animation frame  
• Manages gameplay  
• Converts input to actions  
• Processes NPC behavior  
• Resolves physics  
• Resolves other interactions  
• Updates the scene graph  
• Transforms nodes  
• Enables/disables nodes  

---

Scene Graphs
## The Two Main RootLayer Methods

<table>
<thead>
<tr>
<th>start()</th>
<th>update()</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Handles the game assets</td>
<td>• Called each animation frame</td>
</tr>
<tr>
<td>• Attaches the asset loaders</td>
<td>• Manages gameplay</td>
</tr>
<tr>
<td>• Loads immediate assets</td>
<td>• Converts input actions to actions in root</td>
</tr>
<tr>
<td>• Starts the Scene Graph</td>
<td>• Resolves other interactions</td>
</tr>
<tr>
<td>• Creates any player modes</td>
<td></td>
</tr>
<tr>
<td>• But does not launch yet</td>
<td></td>
</tr>
<tr>
<td>• Waits for assets to load</td>
<td>• Updates the scene graph</td>
</tr>
<tr>
<td>• Like GDXRoot in 3152</td>
<td>• Transforms nodes</td>
</tr>
<tr>
<td></td>
<td>• Enables/disables nodes</td>
</tr>
</tbody>
</table>

**stop()** just cleans up after **start()**

Does not draw! Handled separately
The Cocos2D-x Philosophy

- Each model is its own node
  - Node coords = texture coords
  - Model pos = node pos

- Each node is a Box2D body
  - Scene is a Box2D world
  - Nodes moved automatically

- Each node processes input
  - Node has custom listener
  - Gets touches on that node

- Massive violation of MVC

Scene Graphs
The Cocos2D-x Philosophy

- Each model is its own node
  - Node coords = texture coords
  - Model pos = node pos

- Each node is a Box2D body
  - Scene is a Box2D world
  - Nodes moved automatically

- Each node processes input
  - Node has custom listener
  - Gets touches on that node

- Massive violation of MVC
The Problem: Physics

2D Physics “Island”

Parallax layers (e.g. beat-em up)
The Problem: Physics

2D Physics “Island”

Separate Island

Parallax layers (e.g. beat-em up)
The Problem: **Physics**

Scene Graphs
The Problem: Physics

How big is that scene graph?
Bigger Problem: Rendering

- Cocos2D 2.x Rendering
  - Node = new graphics call
  - Even if the same texture
  - Many calls on small data

- GPU cards do not like this
  - Each call requires card I/O
  - Want few calls on large data
  - Else performance stalls

- **Result**: Horrible framerate
  - Cocos2D 3.x changed this
What Do We Really Want?

- Sprites = textured triangles
- Gather all sprite vertices
- Make one list of triangles
- Send them to GPU at once
- But stall on texture change
- Reorder data on texture
- Limits texture switches
- Safe if there is no overlap
- Is there a name for this?
What Do We Really Want?

- Sprites = textured **triangles**
- Gather all sprite vertices
- Make one list of triangles
- Send them to GPU at once

- But stall on texture change
- Reorder data on texture
- Limits texture switches
- Safe if there is **no overlap**

- Is there a name for this?
What Do We Really Want?

- Sprites = textured **triangles**
- Gather all sprite vertices
- Make one list of triangles
- Send them to GPU at once
- But stall on texture change
  - Reorder data on texture
  - Limits texture switches
  - Safe if there is **no overlap**
- Is there a name for this?
What Do We Really Want?

- Sprites = textured **triangles**
- Gather all sprite vertices
- Make one list of triangles
- Send them to GPU at once
- But stall on texture change
  - Reorder data on texture
  - Limits texture switches
  - Safe if there is **no overlap**
- Is there a name for this?

Scene Graphs
What Do We Really Want?

- Sprites = textured **triangles**
  - Gather all sprite vertices
  - Make one list of triangles
  - Send them to GPU at once
- But stall on texture change
  - Reorder data on texture
  - Limits texture switches
  - Safe if there is **no overlap**
- Is there a name for this?
What Do We Really Want?

- Sprites = textured triangles
  - Gather all sprite vertices
  - Make one list of triangles
  - Send them to GPU at once

- But stall on texture change
  - Reorder data on texture
  - Limits texture switches
  - Safe if there is no overlap

- Is there a name for this?
  - SpriteBatch!
So Why Use a Scene Graph?

- **Animation** is much easier!
- Can reduce filmstrips
  - Break asset into parts
  - Each has a coord system
  - Transform each separately
- Decouple animation loop
  - Update does not set frame
  - Node advances frame
  - Update switches animations
- **Exception**: ShipDemo
So Why Use a Scene Graph?

- **Animation** is much easier!
- Can reduce filmstrips
  - Break asset into parts
  - Each has a coord system
  - Transform each separately
- Decouple animation loop
  - Update does not set frame
  - Node advances frame
  - Update switches animations
- **Exception**: ShipDemo
So Why Use a Scene Graph?

- **Animation** is much easier!
- Can reduce filmstrips
  - Break asset into parts
  - Each has a coord system
  - Transform each separately
- Decouple animation loop
  - Update does not set frame
  - Node advances frame
  - Update switches animations
- **Exception**: ShipDemo
So Why Use a Scene Graph?

- **Animation** is much easier!
- Can reduce filmstrips
  - Break asset into parts
  - Each has a coord system
  - Transform each separately
- Decouple animation
  - Update does not set frame
  - Node advances frame
  - Update switches animations
- **Exception**: ShipDemo

See Cocos2D Animation Tutorials
Also Good for Touch Interfaces

- Touch handler requires
  - Which object touched
  - Location inside object
- Scene graph is a *search tree*
  - Check if touch is in parent
  - … then check each child
  - Faster than linear search
- But limit this to a *search*
  - No input control in node
  - Polling over callbacks
Scene Graphs + Performance?

- Decouple scene & renderer
  - **Update** pass modifies scene
  - **Render** pass to SpriteBatch
  - **Draw** pass sends to GPU

- Renderer in second thread?
  - Can draw even update slow
  - Decoupled animation can still look very smooth

- What Cocos2D 3.x does
  - Sort-of (**textures only**)
  - Wireframes are expensive!
Optimizing Performance: \textit{zOrder}

- Can specify draw order
  - Give each child a \textit{z}-value
  - Ties are permitted
  - Objects drawn by \textit{z}-order (ties broken by graph order)

- Controls texture switching
  - One texture = one \textit{z}-value
  - Reduces it to one draw call
  - \textbf{Example}: RocketDemo

- Should do when no overlap
  - Big Cocos2d optimization
Optimizing Performance: Atlases

- **Idea**: Never switch textures
  - Film strip is many images
  - We can draw part of texture
  - One texture for everything?
  - Called a *texture atlas*

- **Disadvantages**?
  - Cannot tile textures
  - Can be tricky to pack

- **Ideal for** interface design
  - Images for UX widgets
  - Often small and compact
Making Custom Graph Nodes

- Demos had **custom nodes**
  - **Filmstrip** (simple animation)
  - **PolygonSprite** (tiled shapes)
- Emulate existing classes
  - **Sprite** if need textures
  - **DrawNode** if no textures
  - See how they are written
- Two main methods needed
  - Static constructor
  - The `draw` method
Making Custom Graph Nodes

- Demos had **custom nodes**
  - **Filmstrip** (simple animation)
  - **PolygonSprite** (tiled shapes)
- Emulate existing classes
  - **Sprite** if need textures
  - **DrawNode** if no textures
  - See how they are written
- Two main methods needed
  - **Static constructor**
  - The **draw** method

In future lectures. See code samples.
The Draw Command

```c
void PolygonNode::draw(Renderer *renderer,
    const Mat4 &transform, uint32_t flags) {

    // Don't calculate culling if transform was not updated
    _insideBounds = (flags & FLAGS_TRANSFORM_DIRTY) ?
        renderer->checkVisibility(transform, _contentSize) :
        _insideBounds;

    if (_insideBounds) {
        _command.init(_globalZOrder, _texture->getName(),
            getGLProgramState(), _blendFunc, *_triangles, transform);
        renderer->addCommand(&_command);
    }
}
```

Scene Graphs
The Draw Command

```cpp
void PolygonNode::draw(Renderer *renderer, 
const Mat4 &transform, uint32_t flags) {

    // Don't calculate culling if transform was not updated
    _insideBounds = (flags & FLAGS_TRANSFORM_DIRTY) ? 
        renderer->checkVisibility(transform, _contentSize) :
        _insideBounds;

    if (_insideBounds) {
        _command.init(_globalZOrder, _texture->getName(), 
            getGLProgramState(), _blendFunc, *_triangles, 
            transform);
        renderer->addCommand(&_command);
    }
}
```
Cocos2D Command Types

- **QuadCommand**
  - Can draw a textured rectangle
  - Batches quads together if possible
  - Used by the Sprite

- **TrianglesCommand**
  - Batched together, but not with quads
  - Used by PolygonNode (and others)

- **CustomCommand**
  - Give it a callback function
  - Can execute arbitrary OpenGL
Using CustomCommand

```cpp
void DrawNode::draw(Renderer *renderer, const Mat4 &transform,
                     uint32_t flags) {

    _customCommand.init(_globalZOrder);
    _customCommand.func =
        CC_CALLBACK_0(DrawNode::onDraw, this, transform,
                      flags);
    renderer->addCommand(&_customCommand);
}

void DrawNode::onDraw(const Mat4 &transform, uint32_t flags) {
    auto glProgram = getGLProgram();
    // OpenGL Code
    ...
}
```

Scene Graphs
void DrawNode::draw(Renderer *renderer, const Mat4 &transform, uint32_t flags) {
    _customCommand.init(_globalZOrder);
    _customCommand.func =
        CC_CALLBACK_0(DrawNode::onDraw, this, transform, flags);
    renderer->addCommand(&_customCommand);
}

void DrawNode::onDraw(const Mat4 &transform, uint32_t flags) {
    auto glProgram = getGLProgram();
    // OpenGL Code
    ...}

Draws triangle vertices created by helpers.
Using CustomCommand

```cpp
void DrawNode::draw(Renderer *renderer, const Mat4 &transform, uint32_t flags) {
    _customCommand.init(_globalZOrder);
    _customCommand.func =
        CC_CALLBACK_0(DrawNode::onDraw, this, transform, flags);
    renderer->addCommand(&_customCommand);
}

void DrawNode::onDraw(const Mat4 &transform, uint32_t flags) {
    auto glProgram = getGLProgram();
    // OpenGL Code
    ...
}
```

You can probably subclass and just add new helpers

Draws triangle vertices created by helpers.

Scene Graphs
Summary

- Cocos2D uses **scene graphs** to draw
  - Tree of nodes with relative coordinate systems
  - Root node, Layer, processes all touch input

- Cocos2D has integrated **too much** into graph
  - Physics should be separate (RocketDemo)
  - Touch is useful, but separate listeners from node

- Cocos2D 3.x has made major improvements
  - Sprites and polygons are now batched together
  - But need to understand how to optimize