Lecture 11

Scene Graphs
Structure of a Cocos2D-x Application

Deployed Platform → App Delegate → Director → Scene → Scene → Scene
Structure of a Cocos2D-x Application

Deployed Platform

App Delegate

iOS or Android

Memory policy (last lecture)

Director

Scene

Scene

Scene
Structure of a Cocos2D-x Application

- Deployed Platform
- App Delegate
- Director
  - Scene
  - Scene
  - Scene

Access as a singleton

Director::getInstance()
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

Custom code must go here
Aside: When Do We Load Assets?

- Deployed Platform
- App Delegate
- Application Start-up
- Director
- Scene
- Scene
- Scene

Choice affects **design** & **ownership** of the asset manager.
Or any subclass

Necessary to support touch input
Each Node is a Coordinate System

- Bounded box inside
- Coords relative to parent box
- Device/Screen Coordinates

Scene Graphs
Each Node is a Coordinate System
Each Node is a Coordinate System

Layer

Node

Node

Node

Node

Scene Graphs
Each Node is a Coordinate System

Layer

Node

Node

Node

Node

Node

Node

Origin

Origin

Origin

Origin

Scene Graphs
Settings Pass Down the Graph

Layer

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

Layer

Node

Node

Node

Disabling the parent also disables children
The Scene Graph

Necessary to support touch input
class GameMode: public Layer {
private:
  // Internal controllers/models
...
public:
  /**
   * Creates scene for game mode
   */
  static Scene* createScene();
  // Custom gameplay methods
  ...
};

Scene* GameMode::createScene() {
  // 'scene' is an autorelease object
  auto scene = Scene::create();
  // 'layer' is an autorelease object
  auto layer = GameMode::create();
  // add layer as a child to scene
  scene->addChild(layer);
  // return the scene
  return scene;
  }
  // Custom gameplay methods
  ...

Scene Graphs
# The Two Main Layer Methods

<table>
<thead>
<tr>
<th><strong>init()</strong></th>
<th><strong>update()</strong></th>
</tr>
</thead>
</table>
| - Initializes the models  
  - Including physics bodies  
- Initializes subcontrollers  
- Initializes the scene graph  
  - Last-minute asset loading  
  - Places nodes in the tree  
- Registers input listeners  
- **STARTS** update | - Polls the user input  
- Manages gameplay  
  - Converts input to actions  
  - Processes NPC behavior  
  - Resolves physics  
  - Resolves other interactions  
- Updates the scene graph  
  - Transforms nodes  
  - Enables/disables nodes |
The Two Main Layer Methods

<table>
<thead>
<tr>
<th>init()</th>
<th>update()</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Initializes the models</td>
<td>- Polls the user input</td>
</tr>
<tr>
<td>- Including physics bodies</td>
<td>- Manages gameplay</td>
</tr>
<tr>
<td>- Initializes subcontrollers</td>
<td>- Converts input actions to actions for the current node or node group</td>
</tr>
<tr>
<td>- Initializes the scene graph</td>
<td>- Resolves other interactions</td>
</tr>
<tr>
<td>- Last-minute asset loading</td>
<td>- Updates the scene graph</td>
</tr>
<tr>
<td>- Places nodes in the tree</td>
<td>- Transforms nodes</td>
</tr>
<tr>
<td>- Registers input listeners</td>
<td>- Enables/disables nodes</td>
</tr>
<tr>
<td>- STARTS update</td>
<td></td>
</tr>
</tbody>
</table>

*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
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
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?

Scene Graphs
What Do We Really Want?

- Sprites = textured \textit{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 \textit{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?
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: \textbf{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

All one graph node

Discussed last time. See PolygonSprite.
The Draw Command

```cpp
void PolygonSprite::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 PolygonSprite::draw(Renderer *renderer,
const Mat4 &transform, uint32_t flags) {
    _insideBounds = (flags & FLAGS_TRANSFORM_DIRTY) ?
            renderer->checkVisibility(transform, _contentSize) :
            _insideBounds;
    if (_insideBounds) {
        _command.init(_globalZOrder, _texture->getName(),
            getGLProgramState(), _blendFunc, *_triangles, transform);
        renderer->addCommand(&_command);
    }
}
```

Check if on screen

Cocos2d Class

OpenGL Parameters

Send command to renderer

Scene Graphs
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 `PolygonSprite`

- **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
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
    ...
}
```

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.
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