4 Overview on Approaches to Multimedia Programming

4.1 Historical Roots of Multimedia Programming

4.2 Squeak and Smalltalk: An Alternative Vision

4.3 Frameworks for Multimedia Programming

Overview of Java Media APIs
- Architectures for Extended Graphics APIs
- Examples for Java Animation APIs
  - JGoodies, SceneBeans, Piccolo

4.4 Further Approaches & Systematic Overview

Java Media APIs

- Java was from its beginnings intended as a multimedia programming language:
  - “Oak”, Java’s predecessor: designed to control Set Top Boxes for Interactive TV
- Java Media APIs
  - Loose collection of APIs defined and maintained by Sun
  - Main APIs: Advanced Imaging (JAI), Java Media Framework (JMF), Java 3D
  - APIs which have become part of standard distribution: Java 2D, Java Sound
- Style rather heterogeneous
- Not all multimedia programming tasks covered
  - E.g. animation
  - “Unofficial” APIs and implementations try to fill the gap
Summary on Java Media APIs

- Main application areas:
  - Creation of media creation and editing software
  - Not targeted for individual creation of multimedia applications
- Architectural principles:
  - Processing chains
  - Prefabricated components for dealing with complex media types (e.g. video)
  - Realized by various software design patterns
    - Strategy objects encapsulating e.g. a single filter function
    - Pipeline architectures
    - Event handling for synchronisation
- Programming style:
  - Low-level, rather tedious, many technical details
- Expressive power:
  - Very high power when using very low level description (e.g. sound synthesis)
  - Limited power when using pre-fabricated media-processing components
4 Overview on Approaches to Multimedia Programming

4.1 Historical Roots of Multimedia Programming

4.2 Squeak and Smalltalk: An Alternative Vision

4.3 Frameworks for Multimedia Programming

- Overview of Java Media APIs
- Architectures for Extended Graphics APIs
- Examples for Java Animation APIs
  - JGoodies, SceneBeans, Piccolo

4.4 Further Approaches & Systematic Overview

Literature:

Vector Animation Framework

- Media types:
  - Still 2D images
  - Moving 2D images
  - Sound
  - 3D Scenes

- Media types include:
  - Moving 2D images:
    - vector graphics
    - sampled
  - Sound:
    - MIDI
    - sampled

Playback, Create, Process

Examples (Non-official, not widespread!):
- JGoodies Animation (www.jgoodies.com)
- SceneBeans (www.dse.doc.ic.ac.uk/Software/SceneBeans)
- Piccolo & Jazz (www.cs.umd.edu/hcil/piccolo/)
How to Design an Interaction/Animation Framework for Vector Graphics?

• Key concepts needed:
  – Time-dependency: clocks, timers
  – New variants of graphics objects
    » Dynamic, behaviour
• Basic design idea:
  – Graph of objects rendered in a time-dependent way
• How to integrate time- and interaction-dependent behaviour?
  – (Swing) layout + global structured timeline (similar to SMIL)
    » “Time containers”, composed at compile time by method calls
    -> e.g. in JGoodies Animation
  – Scene graphs with local time-dependent interpolators (similar to VRML)
    -> e.g. in SceneBeans
• Please note the analogy to timeline-based vs. script-based animation in Flash!

4 Overview on Approaches to Multimedia Programming

4.1 Historical Roots of Multimedia Programming
4.2 Squeak and Smalltalk: An Alternative Vision
4.3 Frameworks for Multimedia Programming
  Overview of Java Media APIs
  Architectures for Extended Graphics APIs
  Examples for Java Animation APIs
    JGoodies, SceneBeans, Piccolo
4.4 Further Approaches & Systematic Overview
**JGoodies: General Facts**

- This is NOT an official Java standard!
  - Private initiative (Karsten Lentzsch)
  - https://animation.dev.java.net/
  - Here only used as example
- “The framework uses concepts and notions as described by the W3C specification for the Synchronized Multimedia Integration Language (SMIL). Unlike SMIL we use Java to describe the animations - not XML.”
- Similar approach:
  - Chet Haase: https://timingframework.dev.java.net/

**JGoodies Example (1)**

```java
private Animation createAnimation() {
    Animation welcome =
        BasicTextAnimation.defaultFade(
            label1,
            2500,
            "Welcome To",
            Color.darkGray);

    Animation theJGoodiesAnimation =
        BasicTextAnimation.defaultFade(
            label1,
            3000,
            "The JGoodies Animation",
            Color.darkGray);

    Animation description =
        BasicTextAnimations.defaultFade(
            label1,
            label2,
            2000,
            -100,
            "An open source framework|"
            +
            "for time-based|real-time animations|in Java.",
            Color.darkGray); ...
} ...
```
JGoodies Example (2)

Animation all =
  Animations.sequential(new Animation[] {
    Animations.pause(1000),
    welcome,
    Animations.pause(1000),
    theJGoodiesAnimation,
    Animations.pause(1000),
    description,
    Animations.pause(1000),
    features,
    Animations.pause(1000),
    featureList,
    Animations.pause(1500),
  });

JGoodies Example (3)

In main program:
  JFrame frame = new JFrame();
  Intropage ip = new IntroPage();
  frame.setContentPane(ip.build());
  frame.setSize(350, 150);
  ...
  frame.setVisible(true);
  Animator animator =
    new Animator(introPage.animation(), fps);
  animator.start();

Method build() of class IntroPage
  label1 = new BasicTextLabel(" ");
  label1.setFont(font);
  label1.setBounds(0, 0, 350, 100);
  label1.setOpaque(false);
  label2 = ...
  animation = createAnimation();
Methods of JGoodies “Animations” Class

- Offset
  - beginTime
- Parallel
- Pause
  - duration
- Repeat
- Reverse
- Sequential

SceneBeans: General Facts

- This is NOT an official Java standard!
  - Research product from Imperial College, UK
  - http://www-dse.doc.ic.ac.uk/Software/SceneBeans/
  - Animation of formal models of concurrent processes
- Nat Pryce and Jeff Magee: SceneBeans: A Component-Based Animation Framework for Java (at above Web address)
- Software “Animator” interpreting an XML file
Scene Beans

- SceneBeans defines a graphical display using a "scene graph".
  - A Java Bean is a simple software component in Java following naming conventions to enable manipulation in authoring systems.

![Diagram of a scene graph]

Material on SceneBeans adapted from Nat Bryce

Example: Spinning Square

![Example of a spinning square]

P Rectangle
Example: Spinning Square

Example: Spinning Square
Example: Spinning Square

- `T` Rotate
- `T` Translate
- `S` RGBAColor
- `P` Rectangle

Example: Spinning Square

- `Loop`
- `T` Rotate
- `T` Translate
- `S` RGBAColor
- `P` Rectangle
Composable Animations in SceneBeans

An Animation object encapsulates a scene graph and the behaviours that animate it.

SceneGraph
  ↓
  Composite
  ↓
  Activity
  ↓
  Animation
  ↓
  ActivityRunner

Can itself be embedded in a scene graph and run as an activity.

Animation objects are the units of animation design and reuse.

Scene Graph Example (in XML) (1)

```xml
<?xml version="1.0"?>

<animation width="256" height="256">
  <behaviour id="rotor-spin" algorithm="Loop"
    state="${rotor_state=stopped}"
    <param name="from" value="0.0" />
    <param name="to" value="2*pi" />
    <param name="duration" value="1.0" />
  </behaviour>

  <command name="start">
    <start behaviour="rotor-spin" />
  </command>

  <command name="stop">
    <stop behaviour="rotor-spin" />
  </command>

  ...
```

...
Scene Graph Example (in XML) (2)

...<define id="rotor">
  <primitive type="polygon">
    <param name="pointCount" value="4" />
    <param name="points" index="0" value="(0,0)" />
    <param name="points" index="1" value="(-16,96)" />
    <param name="points" index="2" value="(0,100)" />
    <param name="points" index="3" value="(16,96)" />
  </primitive>
</define>

<define id="rotors">
  <style type="RGBAColor">
    <param name="color" value="000000" />
    <primitive type="circle">
      <param name="radius" value="12" />
    </primitive>
  </style>
  <transform type="rotate">
    <param name="angle" value="1.0" />
    <animate param="angle" behaviour="rotor-spin" />
  </transform>
  <transform type="rotate">
    <param name="angle" value="2*pi/3" />
    <paste object="rotor" />
  </transform>
  <transform type="rotate">
    <param name="angle" value="4*pi/3" />
    <paste object="rotor" />
  </transform>
</define>

Scene Graph Example (in XML) (3)

...<transform type="rotate">
  <param name="angle" value="1.0" />
  <animate param="angle" behaviour="rotor-spin" />
...<transform type="rotate">
  <param name="angle" value="2*pi/3" />
  <paste object="rotor" />
</transform>
<transform type="rotate">
  <param name="angle" value="4*pi/3" />
  <paste object="rotor" />
</transform>
</define>
<draw>
  <paste object="rotors" /> .......
University of Maryland “Piccolo” Framework

- “A revolutionary way to create robust, full-featured graphical applications in Java and C#, with striking visual effects such as zooming, animation and multiple representations.”
  - Piccolo is a layer built on top of a lower level graphics API.
  - PiccoloJava is written in 100% java, and is based on the Java2D API.
  - Piccolo uses a “scene graph” model, this means that Piccolo keeps a hierarchical structure of objects and cameras.
- “History”:
  - Ken Perlin, New York University: “Pad” zoomable interface
  - Ben Bederson, Jim Hollan, Bellcore: “Pad++”
  - Ben Bederson et al, UMD: Jazz
    » Many objects
  - Ben Bederson, Jesse Grosjean, UMD: Piccolo
- Also NOT an official Java standard but in widespread use

Monolithic and Polylithic Class Hierarchies

- **Monolithic**: Primarily uses compile-time inheritance to structure and extend functionality
- **Polylithic**: Primarily uses run-time composition to structure and extend functionality
  - More flexible, but creation of MANY objects
Piccolo Terminology

- **PNode:** Any object that wants to paint itself on the screen should inherit from the node class. In addition to painting on the screen all nodes may have other "child" nodes added to them.
- **PCamera:** Cameras are nodes that have an additional view transform and a collection of layers.
- **PLayer:** Layer nodes are nodes that can be viewed by one or more cameras. They maintain a list of the cameras that are viewing them, and notify these cameras when they are repainted.
- **PRoot:** The PRoot serves as the topmost node in the Piccolo runtime structure.
- **PCanvas:** The PCanvas views the scene graph through a PCamera. It forwards input events to that camera, and uses that camera to draw itself.

Activities in Piccolo

- Activities control some time-dependent aspect of the Piccolo system, usually some part of a node.
- This behavior may be of fixed duration or may continue until some termination condition is met (or perhaps forever).
- Activities are scheduled by the PRoot until they have completed.
- Each activity has a start time and a duration, that together determine when an activity starts stepping and how long it continues to step.
  - **PActivity public PActivity**
    
    ```java
    (long aDuration, long aStepRate,long aStartTime)
    ```
  - **aDuration:** –1 for infinite
  - **aStepRate:** the maximum rate that this activity should receive step events
  - **aStartTime:** the time (relative to System.currentTimeMillis()) that this activity should start
- **protected void activityStep(long elapsedTime)**
  - Execution of activity
Example: Animation in Piccolo (1)

```java
package edu.umd.cs.piccolo.tutorial;
import java.awt.Color;
import edu.umd.cs.piccolo.*;
import edu.umd.cs.piccolo.activities.*;
import edu.umd.cs.piccolo.nodes.*;
import edu.umd.cs.piccolox.*;

public class EffectsFrame extends PFrame {
    public void initialize() {
        // Create the Target for our Activities.
        // Create a new node that we will apply different activities to, place that node at location 200, 200.
        final PNode aNode =
            PPath.createRectangle(0, 0, 100, 80);
        PLayer layer = getCanvas().getLayer();
        layer.addChild(aNode);
        aNode.setOffset(200, 200);
        ...
    }
}
```

Example: Animation in Piccolo (2)

```java
... // Extend PActivity.
// Store the current time in milliseconds for use below.
long currentTime = System.currentTimeMillis();

// Create a new custom "flash" activity.
PActivity flash =
    new PActivity(-1, 500, currentTime + 5000) {
        boolean fRed = true;

        protected void activityStep(long elapsedTime) {
            super.activityStep(elapsedTime);
            if (fRed) {
                aNode.setPaint(Color.red);
            } else {
                aNode.setPaint(Color.green);
            }
            fRed = !fRed;
        }
    };
...
Example: Animation in Piccolo (3)

```java
getCanvas().getRoot().addActivity(flash); // Schedule it

PActivity a1 =
aNode.animateToPositionScaleRotation
    (0, 0, 0.5, 0, 5000);

PActivity a2 =
aNode.animateToPositionScaleRotation
    (100, 0, 1.5, Math.toRadians(110), 5000);

PActivity a3 =
aNode.animateToPositionScaleRotation
    (200, 100, 1, 0, 5000);

    a1.setStartTime(currentTime); // Schedule it
    a2.startAfter(a1); // Schedule it
    a3.startAfter(a2); // Schedule it

public static void main(String[] args) {
    new EffectsFrame();
}
```

Example Applications Realized with Piccolo (1)

- HiSee: Visualization of multi-dimensional data
- Shrimp: Multiperspective Visualization
Example Applications Realized with Piccolo (2)

SimBrain: Neural network simulation (using also HiSee)