4 Overview on Approaches to Multimedia Programming

4.1 History of Multimedia Programming
4.2 Squeak and Smalltalk: An Alternative Vision
4.3 Director and Lingo: Advanced Multimedia Authoring
4.4 Frameworks for Multimedia Programming
   - Overview of Java Media APIs
   - Architectures for Extended Graphics APIs
   - Examples for Java Animation APIs
     JGoodies, SceneBeans, Piccolo
4.5 Summary and Trends
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
Java Media APIs

Java 2D
- Still 2D images: includes
- Moving 2D images: includes
- Vector graphics
- Sampled

Java Sound
- Sound
- MIDI
- Sampled

Java 3D
- 3D Scenes

Java Media Framework
- Media types
- Playback
- Create
- Process

Includes:
- 2D images
- Moving 2D images
- Vector graphics
- Sampled
- MIDI
- Sampled
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 History of Multimedia Programming
4.2 Squeak and Smalltalk: An Alternative Vision
4.3 Director and Lingo: Advanced Multimedia Authoring
4.4 Frameworks for Multimedia Programming

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

Literature:
Vector Animation Framework

Media types
- Still 2D images
  - vector graphics
  - sampled
- Moving 2D images
  - vector graphics
  - sampled
- Sound
  - MIDI
  - sampled
- 3D Scenes

Example (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 History of Multimedia Programming
4.2 Squeak and Smalltalk: An Alternative Vision
4.3 Director and Lingo: Advanced Multimedia Authoring
4.4 Frameworks for Multimedia Programming
    Overview of Java Media APIs
    Architectures for Extended Graphics APIs
    Examples for Java Animation APIs
        JGoodies, SceneBeans, Piccolo
4.5 Summary and Trends
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\ realtime 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),
    });
Methods of JGoodies “Animations” Class

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

Graphical scenes described by a directed acyclic graph of Java Beans

Leaves of the graph represent primitive shapes, text or images

A scene graph defines a tree of nested coordinate spaces.

Parent nodes modify or compose scenes defined by children

Material on SceneBeans adapted from Nat Bryce
Example: Spinning Square

P Rectangle
Example: Spinning Square
Example: Spinning Square

- **T**: Translate
- **S**: RGBAColor
- **P**: Rectangle

Diagram showing an animation window with a 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
```

Animator

Spinning Square Loop

Rectangle

RGBAColor

Translate

Rotate
Composable Animations in SceneBeans

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

- SceneGraph
  - Composite
    - Activity
      - ActivityRunner

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

Animation objects are the units of animation design and reuse.
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

![Diagram of Monolithic and Polylithic Hierarchies]
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.
  – Piccolo.Java is written in 100% java, and is based on the Java2D API.
  – Piccolo uses a "scenegraph" 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
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 a 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
    (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)

...  // 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();
}
```
4 Overview on Approaches to Multimedia Programming

4.1 History of Multimedia Programming
4.2 Squeak and Smalltalk: An Alternative Vision
4.3 Director and Lingo: Advanced Multimedia Authoring
4.4 Frameworks for Multimedia Programming
4.5 Summary and Trends
Summary

• Development process
  – Iterative development vs. stable design
  – Graphical design vs. program design

• Technologies
  – Either based on programming or on graphical design
  – Integration between both worlds still doubtful
  – Vision:
    » Authoring system for (platform-independent) animations
    » Direct representation in programming language
    » Easy round trip engineering (*Design* -> *Code* -> *Design* -> …)
Trend: Steady Increase of Multimedia Development

• Presentations: Multimedia usage steadily increasing
• Web sites:
  – Presentations required to differ from competitors, elegance
  – Work environment replacing desktop
• Mobility:
  – WLAN, UMTS enable powerful interactive applications for small portable devices
  – User interface required to be simple and independent of keyboard input
• Innovative user interfaces:
  – E.g. VR and AR partially based on multimedia technology
• Visualization of results of complex measurements, simulations, etc.

• There is no “multimedia revolution” but multimedia elements are slowly entering many traditional areas of computing
Trend: Increasing Level of Abstraction in Programming

- Machine language, assembler, high-level programming languages
  - What is the next step?
- Alternatives
  - Authoring tools?
  - Code libraries?
  - Component systems and frameworks?
- Abstract models of multimedia applications
  - Helpful or not?
Various Representations of Same Concept

Component r1 = ...;
Animation frame1 = ...;
Animation frame2 = ...;
Animation all =
    Animations.sequential(
        new Animation[]{
            frame1, frame2});

<layout>
    <region id="r1" ...>
</layout>
<body>
    <seq>
        ...frame1
        ...frame2
    </seq>
</body>

Authoring Tool (Flash-like)
Visions: Provocative Questions

• What is special about multimedia programming?
  – Are there special language concepts?
  – Can multimedia make programming simpler (cf. the Squeak/EToys idea)?

• Will a future multimedia development tool still provide support for a classical, text-based programming language?
  – Is there a way for fully graphic “programming”?
  – If yes, will it be really helpful?

• Will new paradigms supersede the object-oriented one?
  – E.g. “aspects”?
  – Is there a better, more abstract replacement for event handling?

• Which role will be played by abstract models of the underlying platform and of the user interaction itself?
  – Will it ever be possible to develop a multimedia application in a platform-independent way?