UI Toolkits & Graphics
APIs in the year 3000

Josh.Marinacci@palm.com
who am i?

josh.marinacci@palm.com

@joshmarinacci

JoshOnDesign.com
Flying Saucer

The 100% Java XHTML/CSS2 Renderer

What is it?

Flying Saucer is a Java library which lets you embed XHTML inside of your application. Anywhere you use styled text you can use Flying Saucer.

Alice's Adventures in Wonderland

Once considering how in the world she was to get out again.

The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself before she found herself falling down a very deep well.

White Rabbit checking watch

Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next. First, she tried to look down and make out what she was coming to, but it was too dark to see anything: then she looked at the sides of the well, and noticed that they were filled with cupboards and book-shelves; here and there she saw maps and pictures hung upon pegs. She took down a jar from one of the shelves as she passed; it was labelled "ORANGE MARMALADE" but to her great disappointment it was empty: she did not like to drop the jar for fear of killing somebody, so managed to put it into one of the cupboards as she fell past it.

"Well!" thought Alice to herself, "after such a fall as this, I shall think nothing of tumbling down stairs! How
Co-wrote Swing Hacks
Windows Look and Feel
NetBeans GUI builder improvements

The Preview Design button (in the toolbar) enables you to test the design of the form.

Set JButton1's action property using: Default editor

Action: foo
Action's Class: desktopapplication1.DesktopApplication1View
Action's Method: @Action void foo()
Attributes:
- Background Task
- Basic
  - Text: Do Something
  - Tool Tip:
  - Accelerator: Ctrl, Shift, Alt, Meta (Mac only)
  - Letter:
  - Icon: Small Icon, Large Icon

Help Reset to default Cancel OK
Class JApplet

```
java.lang.Object
   |-- java.awt.Component
      |-- java.awt.Container
         |-- java.awt.Panel
         `-- java.applet.Applet

public class JApplet
```

An extended version of java.applet.Applet that adds support for inserting input and painting behavior in front of the applets children (see glassPane), support for special children that are managed by a LayeredPane (see rootPane) and for Swing MenuBars.
JavaDoc: 2010

Overview Package Class Use Tree Deprecated Index Help

java.applet

Class Applet

java.lang.Object
   java.awt.Component
      java.awt.Container
         java.awt.Panel
            java.applet.Applet

All Implemented Interfaces:
   ImageObserver, MenuContainer, Serializable, Accessible

Direct Known Subclasses:
   JApplet

public class Applet
   extends Panel

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

The Applet class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java
public class javafx.scene.shape.ShapeIntersect

Known subclasses: 

Jump to Section: overview variables functions

Overview

A Shape that is the result of the union of all shapes in sequence a intersected with each of the shapes in sequence b. This implementation of Constructive Area Geometry uses the geometric shape of the provided shapes for the Constructive Area Geometry operation. The primitive shape is not affected by many of the node attributes such as fill, stroke, etc.

the code:

```java
import javafx.scene.shape.*;
ShapeIntersect {
    a: Rectangle { width: 100  height: 50 }
    b: Ellipse { centerX: 100  centerY: 25  radiusX: 50  radiusY: 25 }
}
```

produces:
LevelIndicator
Posted by: tounail on: January 5, 2010
In: controls, graphics, real-world, utility • 2 Comments

The LevelIndicator allows you to view a level. Gives it a parameter value "variable" and value "maximum" as reference.

- You can arrange design with CSS
- It has 3 Skins (BAR_VIEW / PERCENT_VIEW / VALUE_VIEW) and you can switch them by clicking on the LevelIndicator

• Looks like:
Client Lead: Java Store
Blogs, Articles, Conferences
In short. I’m a gui guy. I love great user interfaces and I love learning how to make them better.
Rise of Complexity

- more to do
- tools are bigger
- harder / more to learn
- buried under complexity

over time what we want to do on screen becomes more complicated and harder to do, so we throw more hardware at it, and hide it behind easier to use apis.
Higher-level abstractions

- Every phase introduces new higher level abstractions.
- This *has* to happen or we die.

we are moving up the stack and working at a higher level with each progression. Our abstractions are becoming even more abstract. This *has* to happen because you'd never get anything done if you still had to draw the pixels yourselves.

At every step we hide the complexity in an API that someone else has written. Basically, let someone else do the dirty work so I can get back to working on my app. There are always tradeoffs, but those tradeoffs have changed dramatically over time. This happens in every part of software engineering, not just graphics.
History of CG in 60sec
Vector displays: have to know EE stuff and waves and cosines. ick. Still, amazing for the time when all we had was blinking lightbulbs and switches.
Bitmaps: turn pixels on and off. Simple, but slow, and a lot of work to do cool stuff.
C64 & Amiga

C64 & amiga: lots of tricks with video hardware registers to do more than a straight bitmap would do. you have to really know your hardware. extremely low level and hardware specific, but cool effects are possible (for the time)
Offline 3D: very advanced. object oriented, can be easy to use. Insanely slow. hours or days per frame in the late 80s / early 90s.
OpenGL: transforms and transparency implemented in hardware. OpenGL isn't actually 3d. yes it has 3d transforms and matrix operations, but it doesn't really understand 3d objects. It's just about drawing transformed triangles on the screen, over and over again as fast as possible. What's really important is it gives us a baseline and it's hardware agnostic. if something supports OpenGL you can run your app on it. No recoding required.
Voodoo3D

Voodoo3D: computer video games push the state of the art forward. Still hardware specific, but fancy effects are possible, and silicon improvements drive total number of pixels to crazy heights. Gaming created a consumer level arms race that produced rapid speed improvements throughout the 90s and 2000s.
Desktops Still Suck

Desktop OSes are still basically 2D when you aren't running games, with only minimal hardware acceleration of scaled bitmaps and drawing lines.
Scenegraphs: object oriented descriptions of a graphics tree. High level and easy to work with. API hides all hardware details, often not fast though. Lots of research going on to make it faster, though.

Some cool advantages as GPUs get smarter.

Completely hardware agnostic. Possible to make apps for a wide variety of hardware and degrade gracefully.
Retained vs Immediate

- immediate: call your code 60 x per sec
  - faster, more code, lower level
- retained: create shapes and forget them
  - slower, less code, higher level

Around the late 80s and early 90s people began to debate retained vs immediate mode APIs. Hardware was still weak enough that the distinction mattered.

common term: immediate mode vs retained mode
  - immediate: call your drawing code 60 x per second
  - retained: create a rectangle object, let the system handle it for you.

In practice neither technique won out. Most complex applications were a mixture, usually built on a lower level API like OpenGL with the app creating it's own higher level abstractions or using a 3rd party library. Spaghetti code often results.
Finally 2D and 3D are starting to merge with OSX and Vista. The 21st century mixes everything, while continuing the trend to higher level abstractions.
Pixelshaders are everywhere, so common photoshop like effects can be done in hardware and realtime. Drop shadows, fast smooth scaling, video effects.
these big games cost millions of dollars, but most of the cost is in the artwork. creating the entire huge world that you play in. all of it created in special 3d modeling software like Maya and sometime AutoCAD. Then special tools bring this content into the realtime game through conversions.
We are undergoing a further transition today. The GPU is becoming king. Soon all graphics enabled devices will have a GPU in them somewhere, and the GPUs are becoming really good. This means we can do more, but we also have an explosion of devices to work with. Once again higher level abstractions are key.
The Future

- Hardware everywhere, even phones
- Pixelshaders keep evolving
- Highlevel abstractions are needed

everything is going to hardware, even mobile. pixelshader standards are evolving. this makes scene-graphs even more important.
the details of how to do something are abstracted even more.
rock solid animation and faster graphics is possible by uploading geometry to the video card
The GPU from a decade ago or less is now in most smartphones.
Modern Scenegraphs

- OSX: Core Animation
- Windows / Silverlight: XAML
- Flash (mostly)
- Java / JavaFX
- DOM (horrible, but it’s true)
Importing offline objects like Photoshop files and 3D objects used to be the realm of custom software and high-end work. Now it’s moving into everyday tools because we need better ways for developers and designers to work together.

Import offline objects and textures into your codebase and manipulate them directly. General 3D is coming too thanks to new tools and standards for 3D objects.
Immersive Creation

define objects and shapes entirely within the environment
[video or picture of little big world and sketchup and second life]
How do you code for a Holodeck?

perhaps one day you'll define your holodeck interface from within the holodeck interactively with your hands? you might be more of a sculptor than a graphics programmer.
faster
smaller
new inputs: touch, tilt, microphone
visually attractive & highly responsive
GUI != Grid of Rects & Key/Mouse Events
So where does this leave us? In short we need better UI toolkits. Classical UI toolkits like Swing were designed with mid 90s hardware and Win95 as the OS.
Swing

- Most things are rects, avoid shapes & transparency
- Mouse & Keyboard only
- Native themes, not easily skinnable.
- Can’t integrate vector, video, 3d, shaders, etc.
- Not portable or scaleable, No HI-DPI
- Animation, Threading

This means it assumes most things are rects, avoids shapes & transparency because they were slow. Only has events for mouse and keyboard. Not easily skinnable, tons of work put into native themes. Can’t integrate other forms of 2D, Video, 3D, pixel shaders, etc. easily. Not portable. Needs HiDPI. Animation. Threading.

All of these things are *possible* with Swing, but it’s not easy because Swing wasn’t designed with it in mind. Any modern toolkit needs to address these things.
Swing was cross platform in the age of the desktop, but it’s not built for other kinds of devices.
Let's start with the Graphics Stack. First, it needs to be built on a real scenegraph. This means a tree of nodes that can mix shapes and other graphical content with the actual UI controls.

High level, but give you low level access when you need it. Most of the time you can code with high level objects like buttons and shapes, but you can also implement your own draw method when needed.

It should be hardware accelerated but also hardware agnostic, meaning we can fall back on slower implementations as needed. And, as part of hardware acceleration, we should be able to support pixel shader effects for lightning fast shadows, blurs, and transitions, plus new things we haven’t thought of.

Animation should be built in. The beauty of a scenegraph is that just as you declare your scene and have the computer take care of drawing it, you can declare your animation and have the computer take care of that too. You can say things like: move button from here to here and rotate it all over 6 seconds, and go from 0.5 opacity to fully opaque. This can be done in just a couple of lines of code instead of writing a bunch of loops and drawing code by hand.
The toolkit of the future needs to support lots of kinds of inputs, not just mouse and keyboard events. In particular it must support not only touch events, like where the fingers are, but touch gestures. It should let you write code at a high level to look for things like the scroll gesture rather than having to detect if there are multiple fingers moving vertically at a certain threshold.
Skinnable

- No one cares about native themes
- Everything should be skinnable
- Use CSS

For a long time there was a debate in the UI community about toolkits that give you a native look and feel vs toolkits with a custom look. Well, thanks to the web, custom has won. This is not to say you should go off and do everything in a weird way, in fact the feel part such as keyboard shortcuts, *should* try to match the native specs. But the look can be completely custom as long as it looks good. So that means the toolkit must be easily themeable. And today that pretty much means CSS. It’s the defacto way of separating style from content. There’s no need to reinvent the wheel.
Threads are Evil

- Single threaded
- background tasks
- responsiveness tools
- animation handled separately

Originally Swing was partially multi threaded. You could do certain things in a background thread and others only in the foreground. Over time this caused lots of developer confusion and maintenance problems. Now we know that UI toolkits should be single threaded. It makes the code so much simpler to write and avoids lots of currency issues. Of course we still care about responsiveness, so the solution is to include background task support right in the ui. Make it very easy for the developer to explicitly do something in the background, and then come back to the foreground when it’s ready. These aren’t threads. Tasks are a higher level construct which are easily managed by the system, with threading rules enforced. It also helps to have tools which can measure responsiveness and suggest changes. And of course animation doesn’t require threads because it’s done by the toolkit.
To work well in multiple kinds of devices with different screens the toolkit must be scaleable. Since we are built on a vector based scenegraph we pretty much get this for free. We can zoom in and out with everything transformed properly. It also means that an app might be used on multiple devices, so we need to scale parts of our UI to match the apparent resolution of the device, not just the physical DPI. Just look at the iPhone 4 with double the pixels. The answer is to do layout and scaling based on some resolution neutral measurement like Font ems. If everything is based on ems then your UI will always scale to the appropriate size.
Java based

Platform Agnostic Scenegraph: Java2D, GL, SDL

Fluent APIs: Layout using HTML 5 Flex Box

CSS Skinning

BSD License

Java based, so it supports any language the JVM supports

Built on an gfx stack agnostic scenegraph. Java2D, OpenGL, SDL, etc. No AWT exposed. This is especially important for embedded devices where licensing the standard JavaSE implementation is expensive or impossible.

Fluent APIs: layout based on HTML 5 flexbox model and grids

All UI controls are skinned using CSS 3.

Fully open source: BSD licensed for maximum exposure
CSS Skinned Controls

- Simple HBox with buttons
- Button Sizing
- Simple HBox
- Simple HBox with baseline
- Simple HBox Bottom
- HBox with spacers
- VBox simple
- VBox simple, right aligned
- VBox simple, stretch
- VBox with toolbar and statusbar
- Complex center aligned dialog
- Grid Test: New doc dialog
- Controls baseline test
- Textbox Grid Width
- HBox shrink to fit 1
- HBox explicit preferred width
- HBox explicit preferred height
- HBox explicit pref w & h
- VBox shrink to fit 1
- VBox explicit preferred width
- VBox explicit preferred height
- VBox explicit pref w & h

Stage

- Appearance: Blue
  For the overall look of buttons, menus, and windows
- Highlight Color: Blue
  For selected text
- Place scroll arrows: Together
- Click in the scroll bar to:
  - Jump to the next page
  - Jump to the spot that's clicked
  - Use smooth scrolling
  - Double-click a window's title bar to minimize
- Number of recent items:
  - 10 Applications
  - 10 Documents
  - 10 Servers
  - Use LCD font smoothing when available

Turn off font smoothing for font sizes 4 and smaller
2D / 3D Mixing
Looking for Help

- Overhaul event system
- Design a new CSS theme
- Scenegraph improvements
- More UI controls
- More GFX backends (porting to other devices?)
- Redo background tasks
Conclusion

- Better & portable hardware
- New kinds of inputs
- Visually rich
- UI Toolkits must adapt or be replaced
Amped Hackathon

- Saturday, Sept 25th, @9am
- GTRI Conf Center: 250 14th St.
- Bring a team, Pick a challenge, Win cool prizes
- Free!

http://ampedweb.org/
Info

http://LeonardoSketch.org/


@joshmarinacci

Josh.Marinacci@palm.com