Introduction and Optimizations

Topics Covered

- Part I: What is Nifty GUI?
  - A short Introduction to Nifty
  - Showcase video of usage in games
- Part II: Optimize OpenGL Performance
  - General OpenGL Performance principles
  - How we've applied them in the latest Nifty (1.3.3)
  - Demonstration using JOGL

Part I

What is Nifty GUI?

Live Demo: Nifty 1.2 Tutorial

What is Nifty GUI?

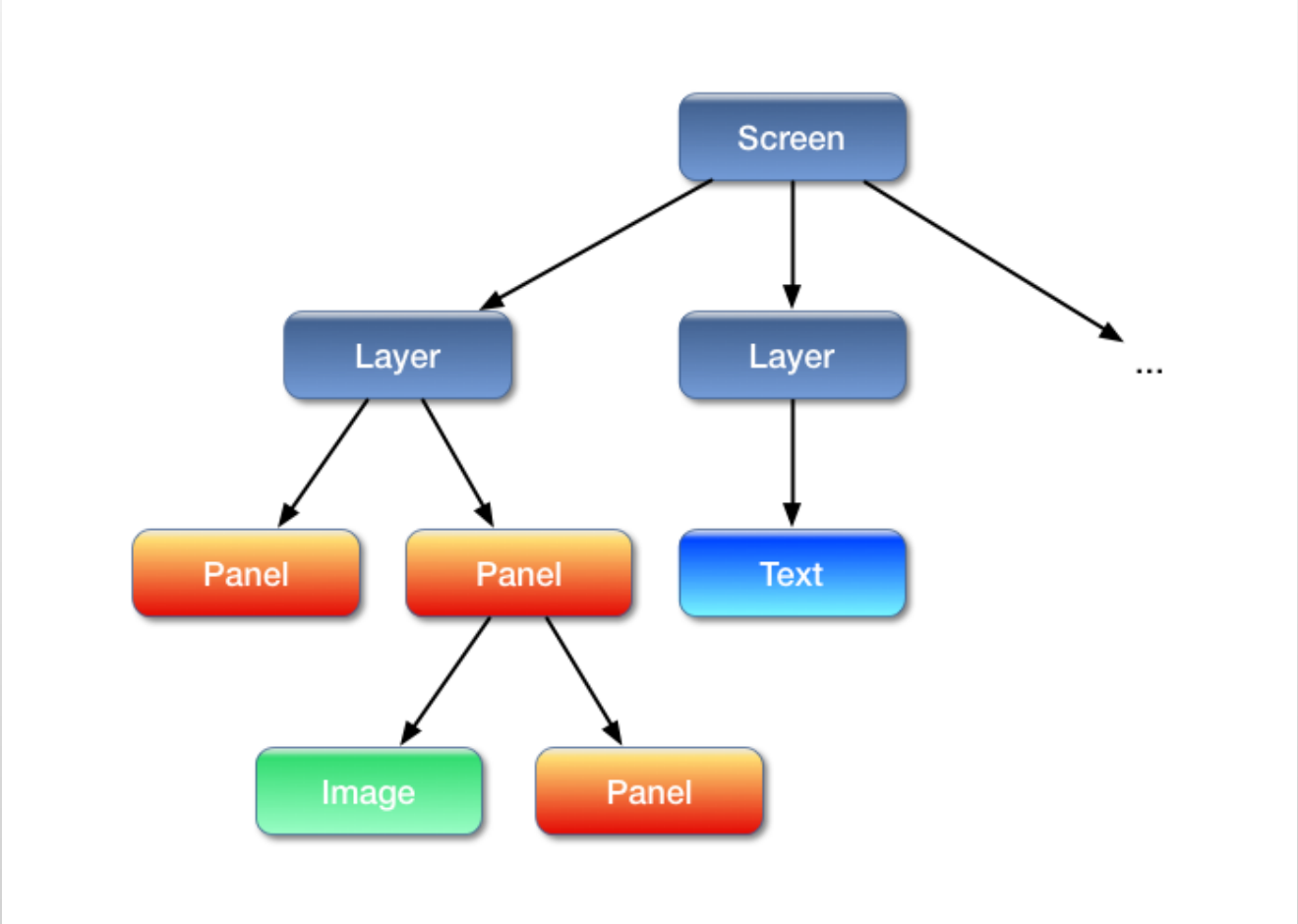
- Java Library in Development since 2007
  - Open source at Sourceforge since April 2008
  - Several major releases so far (latest: 1.3.3)
  - BSD-License
- Use it to build interactive user interfaces for games and other interactive applications
  - Can be extended and styled easily and provides many visual effects
  - However, it is different from AWT/Swing or your other usual GUI framework

What is Nifty GUI?

- Think of Nifty as a scenegraph
  - At it's core Nifty only knows a limited number of core elements:
    - Panel: a rectangle area with a plain color or gradient
    - Image: an Image that can be displayed
    - Text: used to output bitmap fonts
    - Control: A combination of the above to form an abstraction like a Button, Textfield, ListBox and so on
  - Nifty is used to layout and display these elements and handles keyboard and mouse events

What is Nifty GUI?

- Example Scenegraph structure:



What is Nifty GUI?

- Scenegraph is stored in XML
  - XML-Schema (XSD) available for validation and tool support
- Scenegraph can be build from Java as well (using Java Builder Pattern)
- Can be dynamically modified at runtime
  - Add elements
  - Remove elements
  - Move elements

What is Nifty GUI?

- Nifty XML example:

```
<?xml version="1.0" encoding="UTF-8"?>
<nifty>
   <screen id="start">
    <layer id="layer" backgroundColor="#003f" childLayout="center">
      <panel width="50%" height="50%" backgroundImage="nifty-logo-150x150.png"
             imageMode="repeat:0,0,150,150" backgroundColor="#0f08" />
    </layer>
  </screen>
</nifty>
```

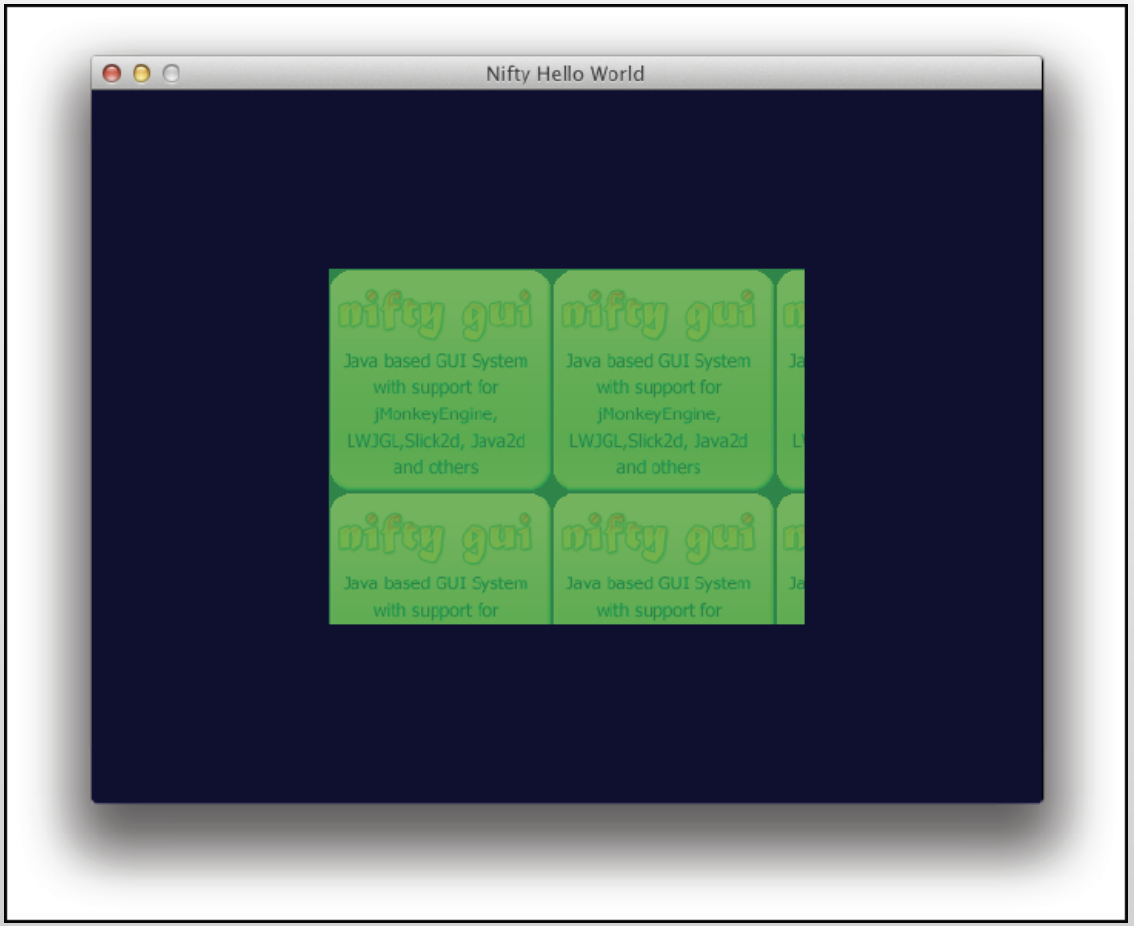
What is Nifty GUI?

- The same example in Java:

```
Screen screen = new ScreenBuilder("start") {{
  layer(new LayerBuilder("layer") {{
    backgroundColor("#003f");
    childLayoutCenter();
    panel(new PanelBuilder() {{
      width("50%");
      height("50%");
      backgroundImage("nifty-logo-150x150.png");
      imageMode("repeat:0,0,150,150");
      backgroundColor("#0f08");
    }});
  }});
}}.build(nifty);
```

What is Nifty GUI?

- Result: Panel with repeated background image



What is Nifty GUI?

- Nifty is very versatile
  - It doesn't rely on a specific graphics framework
  - Existing adapters to several libaries available:
    - JOGL, LWJGL, jMonkeyEngine, Slick2D, Java2D
    - Need a different one? a SPI (Service Provider Interface) is available and can be easily implemented by yourself!
  - Easy to integrate into your code
    - It doesn't take over your complete rendering
    - Just call nifty.render() when you want it to render and nifty.update() to let it update internal state

What is Nifty GUI?

- General use-cases for Nifty:
  - Interactive Menus and Displays
  - Game Option screens
  - In-Game HUD Displays
  - Anything that displays icons or text with Java and wants to do that in a somewhat nifty way ;)
- Let's show some real world examples of Nifty next as its being used in actual games...

Nifty in Games

Video Showcase

Nifty GUI – More Information

- Nifty GUI – The Missing Manual
  - Tutorial and reference documentation
  - Everything you ever wanted to know about Nifty
  - 100+ Pages
  - Available as a free PDF-Download
    - http://sourceforge.net/projects/nifty-gui/files/nifty-gui/1.3.2/nifty-gui-the-manual-1.3.2.pdf/download

Nifty GUI – More Information

- Project Pages:
  - Github:

    https://github.com/void256/nifty-gui

  - Sourceforge:
    https://sourceforge.net/projects/nifty-gui/
  - Twitter:

    https://twitter.com/niftygui

  - Blog:

    http://nifty-gui.lessvoid.com/

Part II

OpenGL Performance Optimization

Explained and Applied with
Nifty GUI 1.x

Overview

- Now we'll go a bit more into details:
  - Introduction to the Nifty Renderer and why it performed not so optimal
  - What can you do in general to speed up OpenGL rendering and how that was applied to Nifty
- Results of the improved Renderer
- Additional benefits of the new way to render

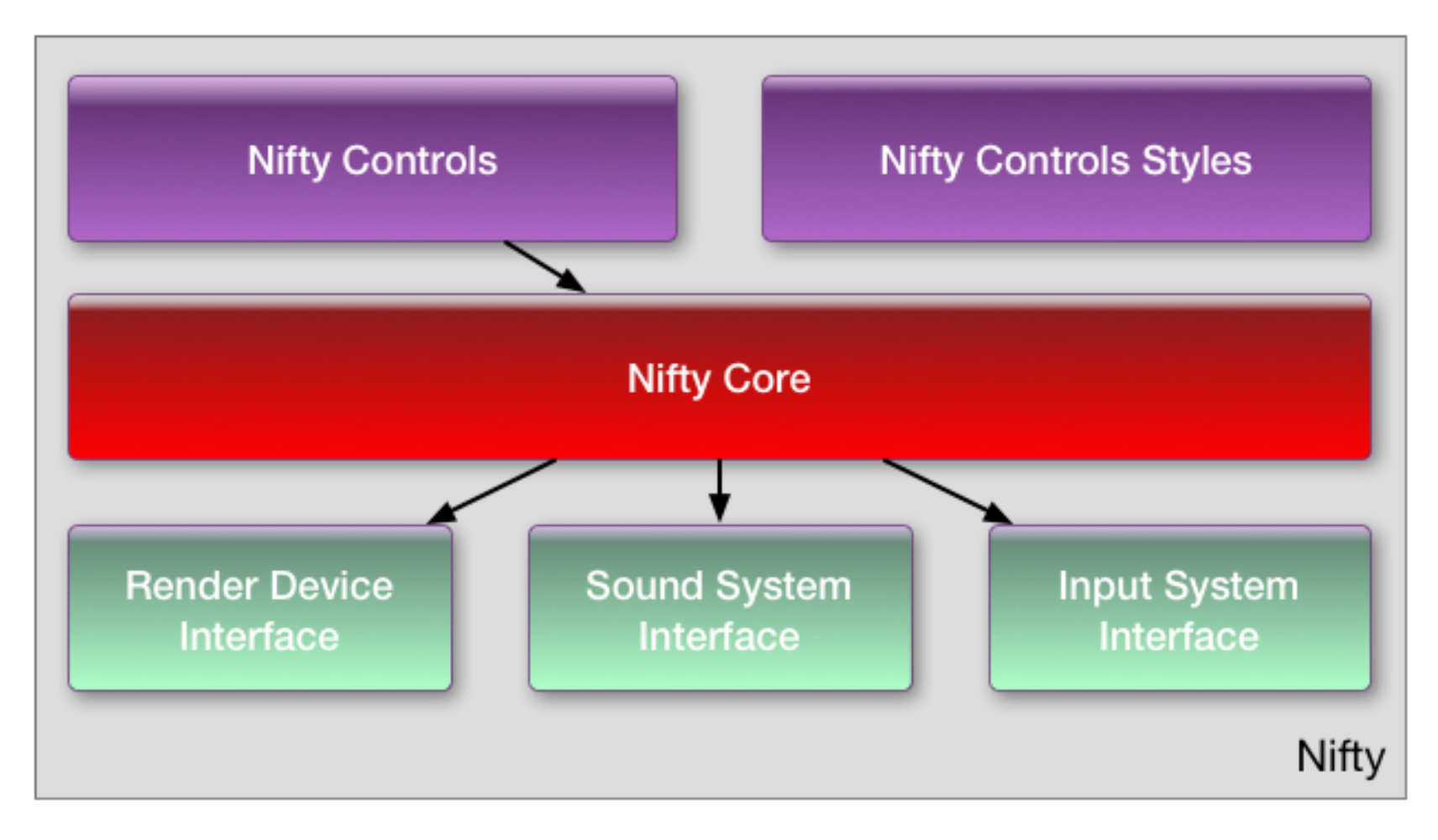
Nifty SPI

Nifty SPI

- Nifty provides Java Interfaces for:
  - Rendering
  - Inputevents (Keyboard + Mouse)
  - Sound output
- Everything Nifty needs is abstracted into these interfaces
- Nifty is build on top of the SPI

Nifty SPI

- Overview of components and the SPI (in green)



Nifty RenderDevice

- A closer look at the RenderDevice SPI:
  - Java interface de.lessvoid.nifty.spi.render.RenderDevice
  - Tasks:
    - provide screen dimensions to Nifty, load images/fonts
    - Methods called when render frame begins and ends
    - Set (OpenGL) states like blend mode and clipping
    - **Main task: render colored quads, images and text**
  - At the end all Nifty-GUI elements end up as quads, images and text

Nifty RenderDevice

- Here are the interesting render*() methods

```
public interface RenderDevice {
...
void renderQuad(int x, int y, int width, int height, Color
color);

void renderQuad(int x, int y, int width, int height, Color
topLeft, Color topRight, Color bottomRight, Color bottomLeft);

void renderImage(RenderImage image, int x, int y, int width,
int height, Color color, float imageScale);

void renderImage(RenderImage image, int x, int y, int w, int h,
int srcX, int srcY, int srcW, int srcH, Color color, float
scale, int centerX, int centerY);

void renderFont(RenderFont font, String text, int x, int y,
Color fontColor, float sizeX, float sizeY);
...
}
```

Nifty RenderDevice

- Typical Nifty rendering looks like this:
  - renderDevice.beginFrame();
    - renderDevice.setBlendMode(BlendMode.BLEND);
    - renderDevice.renderQuad(...);
    - renderDevice.renderImage(..);
    - renderDevice.enableClip(...);
    - renderDevice.renderQuad(...);
    - … and so on
  - renderDevice.endFrame();

So …
What's wrong with the existing implementation?

Rendering – What's wrong

- The current implementation took a somewhat naive, brute-force approach:
  - Vertex submission is using immediate mode with many calls to GL: glBegin, glVertex, …
  - OpenGL state is changed in every render* method
    - Texturing is enabled/disabled all the time and the current texture is switched to different textures
    - clipping rectangle is changed/enabled with glScissors
    - Blendmode is change with glBlendMode as needed

We can do better:
General OpenGL Performance Tips

General OpenGL wisdom

- f.i. found in: „OpenGL Insights, Chapter 25“ or „OpenGL Programming Guide for Mac“:
  - Avoid glBegin/glEnd calls: function call and data copying overhead. Better use VBO or at least client-side vertex arrays.
  - Avoid redundant state changes: Save time by removing unnecessary calls to GL
  - Group primitives together so that they can be rendered with as few draw calls as possible

General OpenGL wisdom

- f.i. found in: „OpenGL Insights, Chapter 25“ or „OpenGL Programming Guide for Mac“:
  - Avoid glBegin/glEnd calls: function call and data copying overhead. Better use VBO or at least client-side vertex arrays.
  - Avoid redundant state changes: Save time by removing unnecessary calls to GL
  - Group primitives together so that they can be rendered with as few draw calls as possible
- So actually, Nifty did that all wrong ;-)

Let's fix the renderer!
One Issue at a time

Step 1: Optimize vertex submission

Step 1: Optimize vertex submission

- Use Vertex Arrays (really a no brainer)
  - Available since early OpenGL 1.1 days (1995)
  - Store all vertices in an array and give OpenGL a pointer to that array
- This helps because:
  - There are no individual glBegin/glVertex/glEnd calls for each quad anymore. This saves us thousands of OpenGL calls for complex GUIs.

Step 1: Optimize vertex submission

- Usage in Nifty:
  - The optimized RenderDevice keeps a single FloatBuffer for all Quads that need to be rendered
  - The render* methods will now simply add four vertices of the current quad to this FloatBuffer
  - In the endFrame method Nifty draws them all with a single glDrawArrays(GL_QUADS...) call
- This reduces the number of OpenGL draw calls significantely

Step 2: Optimize Texture State

Step 2: Optimize Texture State

- Trick: Let texturing enabled all the time so we don't have to switch it always on and off ;)
  - Always submit textured quads
  - How to render a plain colored quad then?
    - Simply render a special prepared area of the texture that contains solid white colored pixels
    - Stretching a single white pixel to the size of the quad will fill the quad with white color
    - Can be combined with vertex colors as well

Step 2: Optimize Texture State

- How to render different images?
  - Combine all individual images into one big texture and keep this texture always active!
- Texture Atlas ftw!
  - solution is simple in theory but was quite involved
  - In most cases you would combine textures as a pre-process but for Nifty it needs to be dynamic
    - Images can be loaded / disposed at any time and are likely to change between Nifty Screens

Step 2: Optimize Texture State

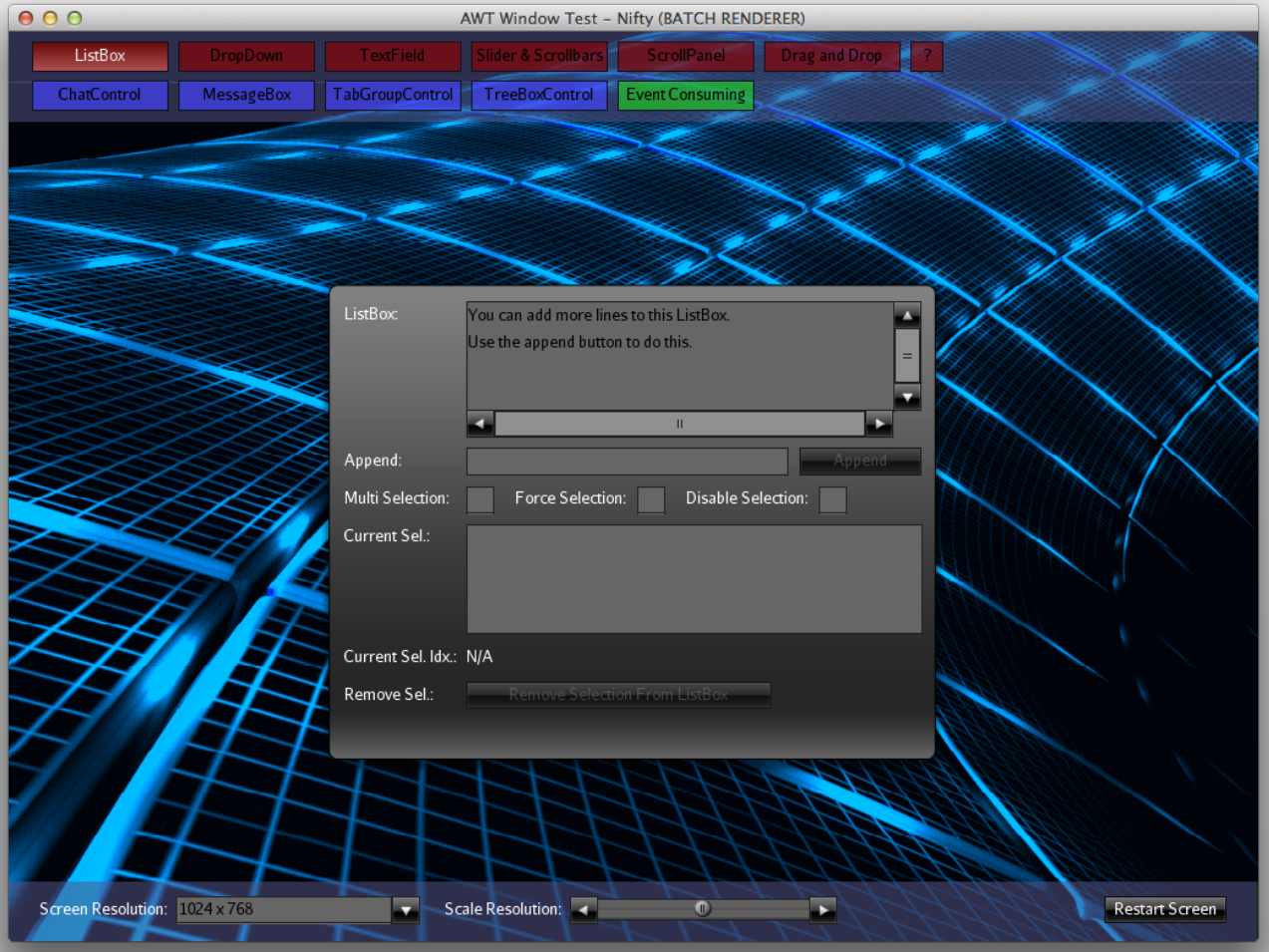
- Let's enter the world of texture packing:
  - Is actually a huge topic
    - see PhD Thesis of Andrea Lodi: „Algorithms for Two Dimensional Bin Packing and Assignment Problems“
  - Looking for a simple solution:
    - Popular „Lightmap Packing Algorithm by Black Pawn“: http://www.blackpawn.com/texts/lightmaps/
    - Java port already available by lukaszdk: https://github.com/lukaszdk/texture-atlas-generator
      - Modified for Nifty to separate algorithm from graphics handling
      - Results available as TextureAtlasGenerator class (single self-contained class in Nifty repo but with no Nifty dependencies)

Step 2: Optimize Texture State

- Nifty texture atlas algorithm overview:
  - Each Nifty Screen starts with an empty texture (2k texture worked well)
  - Nifty tracks which image belongs to which Screen when images are first accessed
  - Position in the texture atlas will be determined by TextureAtlasGenerator algorithm so that all required images are added to the atlas
  - Switching images is then simply a modification of the texture coordinates of individual quads

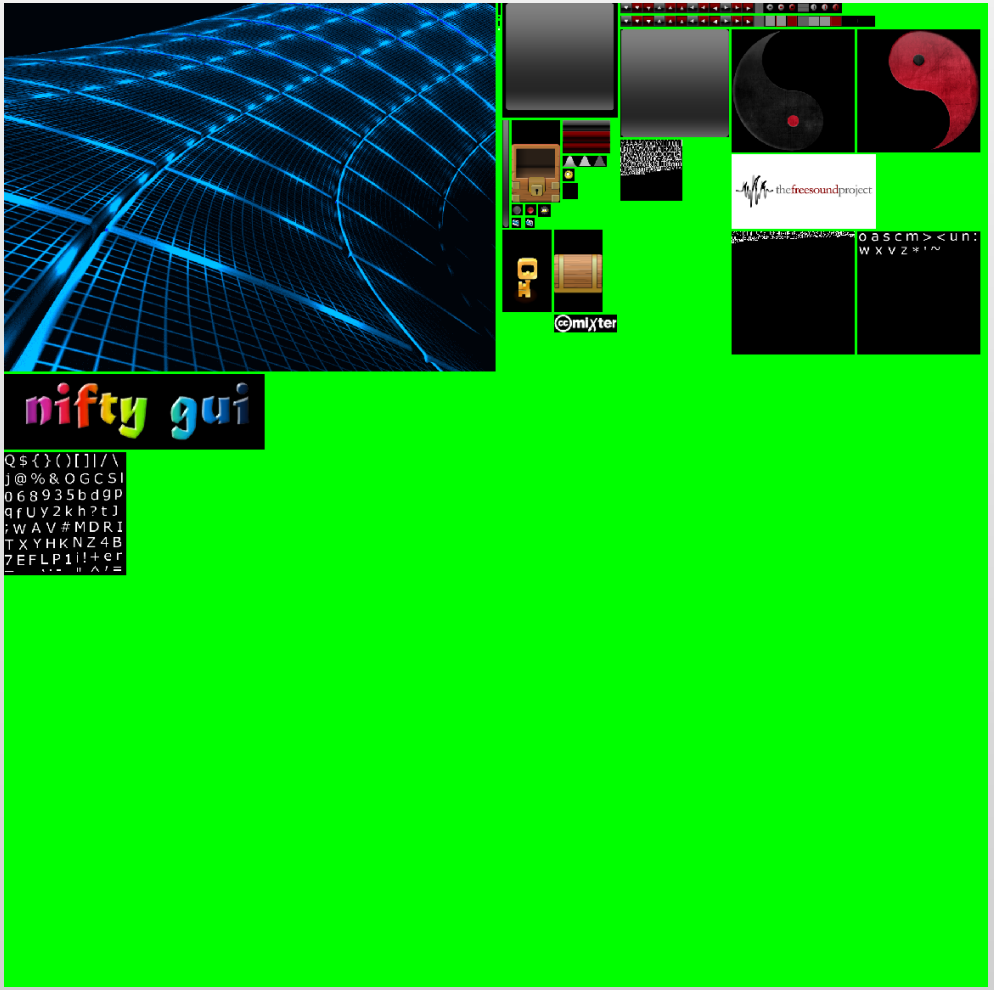
Step 2: Optimize Texture State

- Example: Nifty Standard Controls Demo



Step 2: Optimize Texture State

- On the fly generated texture atlas (2048x2048)



Step 2: Optimize Texture State

- Things to consider:
  - When accessing images while a screen is already active, Nifty needs to upload the texture on the fly
    - glSubImage* is used to update parts of the texture
    - You can get away with a couple of sub texture uploads per frame but would be best if most images are known when the Nifty screen is initialized
  - If the image does not fit into the atlas Nifty will complain in the log but will continue working
  - The texture atlas is reset when you switch from one screen to another

3. Optimize Clipping

3. Optimize Clipping

- Nifty allows you to specify a „childClip“ attribute
  - All child elements will then be clipped to the parent
- glScissors is used for this but can't be used in between a single glDrawArrays call
- Solution: Clip on the CPU!
  - It's only 2d so it's easy
  - Already clipped quads will be added to the vertex array
  - Result: no changes to glScissors necessary!

4. Optimize Blendmode

4. Optimize Blendmode

- Only two blendmodes supported by Nifty:
  - Standard: regular alpha transparency blending
    - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  - Optional blendmode multiply (for special effects):
    - glBlendFunc(GL_DST_COLOR, GL_ZERO)
- Can't change blendmode while rendering VA
- Compromise: Nifty will create a new batch (new vertex array) when blendmode changes
  - Rendering some batches still better than hundreds

Put everything together:
Unified RenderDevice

Unified RenderDevice

- All of the discussed steps required to be supported in all adapter implementations:
  - Batched JOGL, Batched LWJGL, Batched JME3 and so on
  - These implementations have to solve the exact same problems (Texture atlas, Batching, …)
  - Not a very clever approach
- Better solution: solve it once for all adapters:
  - Provide a unified batched RenderDevice impl!

Unified RenderDevice

- Default implementation for the RenderDevice:
  - de.lessvoid.nifty.batch.BatchRenderDevice handles everything we've discussed so far
- Specific adapters still needed but now:
  - Much simpler
  - Will just receive the quads and have to cache them so that they can be rendered in one step later
  - New: need to be able to replace subtextures

Unified RenderDevice

- Additional benefits:
  - Text rendering is now handeled inside of Nifty
    - Each glyph is just a simple quad
    - Kerning, text string width calculations as well as text encoded colors are handled in the same way
    - Text finally looks exactly the same in all libs
- Replacing existing RenderDevice with the new one is very easy
  - Just use the BatchRendererDevice now and provide the BatchRendererDevice with the adapter impl

One more thing...

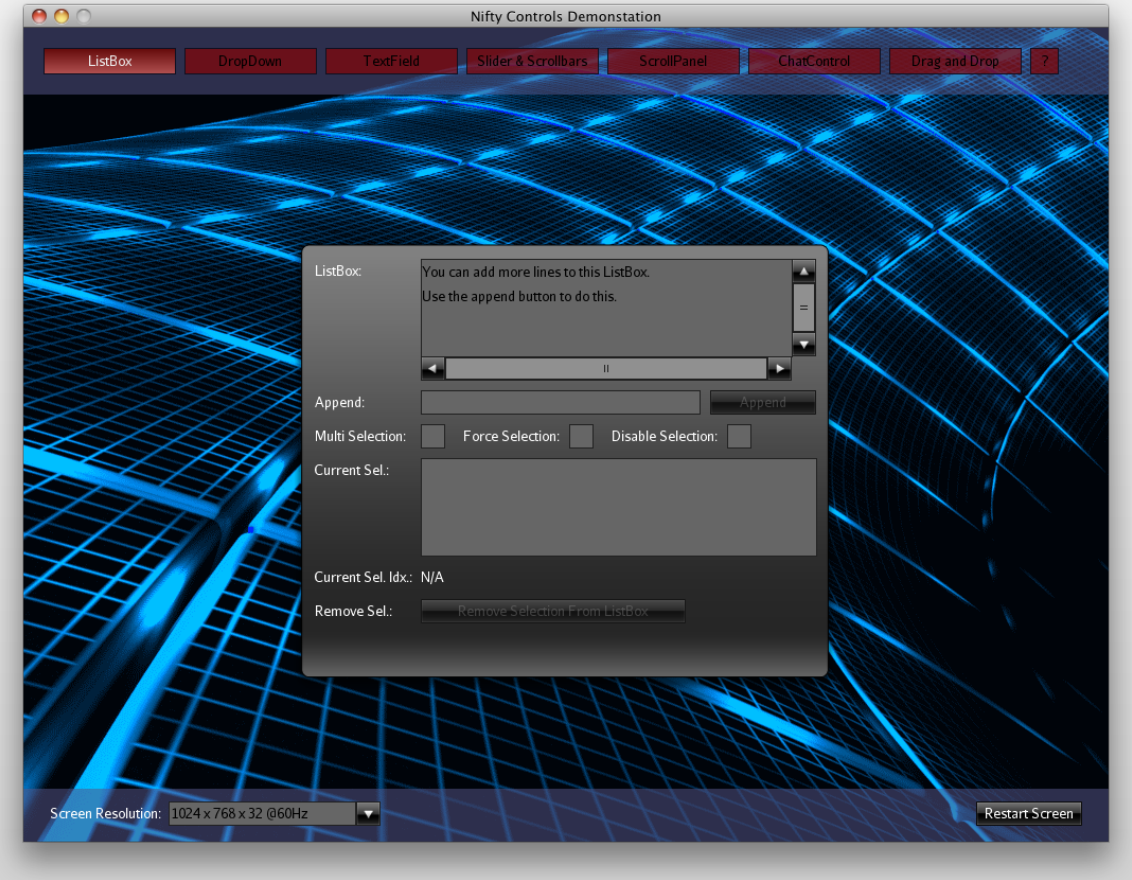
OpenGL Core Profile Support

- Up until now only legacy OpenGL support
  - Couldn't really use Nifty when you use Core Profile
  - Simpler Interface made support now easy
- Can finally use modern OpernGL with Nifty too
- OpenGL ES support using JOGL currently work in progress but should be available soon

Optimization Results

Results of the improved Renderer

- Test: Nifty Standard Controls Demo



Results of the improved Renderer

- Test: Nifty Standard Controls Demo
- Mac Pro, Early 2009, OS X 10.8.4
  - 2 x 2,26 Ghz Quad-Core Intel Xeon
  - 8 GB 1066 Mhz DDR3 ECC
  - ATI Radeon HD 5870 1024 MB
- Results (Rendertime Nifty in ms):

Old Renderer	Batched	Batched Core Profile
1.736 ms	0.617 ms	0.642 ms

Results of the improved Renderer

- How much you gain depends on the complexity of your GUI
  - Very few elements: you don't gain a lot since there is not much overhead
  - Many elements/complex GUI: batched renderer works a lot better (more FPS, less frametime)

Live Test/Demo

Using Controls Demo

Nifty GUI - Contact

- Software Development Jens Hohmuth
  - Commercial Nifty GUI support available
- E-Mail: jens.hohmuth@gmx.de
- Twitter: https://twitter.com/void256