Header Text

# Reflection Idea

- main part of the idea is using dlopen and query of symbols to be able to
  - iterate types
  - iterate member functions
- this has limitation in that it doesn't allow access to variables, however member variables with getter and setter functions will be accessible.

# Quad-Tree Ideas

- Growable Quad-Tree by expanding bounds by adding new root nodes - can just add stream of points with no a-prior of bounds
- Idea is to have nodes with callbacks, so can have trigger areas
- Could have callbacks for when nodes are created or destroyed
- Optimized moving of objects in quad-trees, recursive descent to deepest common branch, then recursive removal and addition from there
- Not a fully formed idea, but perhaps a 2D UI system could benefit from a quad-tree
- Would make it perhaps faster to find a widget a 2D point is in
- Might make it faster to handle re-painting dirty regions

# IDEA

- when switch to using the GPU to accelerate the UI drawing, an idea is to draw
  - textured quads for various pixmap based widget drawing
  - lines for line drawing
  - special quads for text (perhaps using a shader that does SDF or M-SDF etc)
- All these are with screen-facing primitives
- Problem is that order of the calls to add these needs to follow the painters algorithm
- Need to draw them from back to front, but ideally we want to batch the commands
- Problem is if we batch them by type of operation the order is not preserved
- The solution for this is to give the primitives a z value each, and to decrement it each operation
- Then to prevent fore-shortening or distortion, not use a perspective camera, use an ortho camera
- This way now the primitives can be batched and also come out appearing to be draw in the order they were added

Header Text

- Perhaps in some cases for lines or fills with a single color, it could be same shader as for texture, just a 1x1 texture of the color
- Alternative idea is that 2D is no different to 3D and all the UI does is create 3D draw calls
  - the UI in this case can be integrated in to the 3D world or drawn over the top
  - it can be transformed to align with a wall or other elements in the 3D world
  - it doesn't require a seperate render pass to achieve this
  - however how is the Z-order of operations determined with enough precision to avoid z-flighting and allow batching?
- Probably the first idea can allow for more efficient vertex buffer structures and could be transformed in to the world
- how to mix UI in to the world for world space UI?
- separate system for world space UI?
- how about also integration of 3D elements inside the UI? Masking? z-buffer?
- how to reduce the number of render passes? What order to draw them?
- how about alpha? Fake or real? Flag to determine
  - Fake alpha in UI can be done by rendering behind to a buffer, then pre-blending and using that as the background
  - Real alpha is to render behind every frame and always take the overdraw hit
  - Flag used by artist to say if need the blend to update as expecting anything behind to animate/change
  - Possibly a smart UI system could determine the screen areas changed/dirty per frame
  - an optimized set of draw calls could be created to redraw just what is needed

# Other Ideas

- 3D graphics related

- LOD system - based on octrees - objects made of surfaces - a surface is an equation which is passing through octants

- typically triangles are used to represent surfaces - normal artist work flow is with triangles, so needs to be compatible with them

- need to convert the triangles to a surface, possibly can collapse to normals in textures

- then using mesh subdivision via the octants - somehow need to drive triangle subdivision by octant subdivision

Header Text

- some curves / patches can be subdivided arbitrarily.
- Terrain system - similar to LOD system but based on height map in a quadtree instead
- perhaps skew it 60 degrees to make triangles (hexagons)
- still a grid, still 2D, but angled so the grid when split in to two triangle will be two equilateral triangles
- procedurally generated heights, the heights are based on heights at next power of two up and some random seed based on the position
    - therefore it is possible to re-generate the same heights without storing the generated heights
    - it also LODs because you only need to generate according to what is seen to desired resolution
- Path tracing - progressive refinement and reuse what was calculated in previous frames
- Need some kind of data structure that can work on GPU that allows storing not just in 2D framebuffer, but in a 3D structure
- the results from previous frames and that this can be queried on the GPU
- The 2D framebuffer + Z-buffer is kind of like a buffer of 3D points without depth peeling
- Simple idea is to re-project the points from previous frame to current frame
    - either can do it on CPU but need to download and re-upload the framebuffer and need to know the format
    - or can use CUDA or similar
    - or can use a search like SSAO does with a radius and multiple samples per pixel to feel out in the area
        - this technique could search for pixels from previous buffer that closely re-project to current frame
- Color - teal, orange, yellow, green, dash of blue and purple - vivid flattish colors with blobious shapes
- Cool custom font

Header Text

- Cool studio name

# TODO

- Be able to debug from within VIM
- GDB/LLDB with VIM integration - some issues with python version between VIM and plugin
- brew install neovim
- setup ~/.config/nvim/init.vim
- https://github.com/dbgx/lldb.nvim
- curl -fLo ~/.vim/autoload/plug.vim --create-dirs https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
- add "call plug#begin('~/.vim/plugged')" to ~/.vimrc
- Good info: http://blog.rplasil.name/2016/03/how-to-debug-neovim-python-remote-plugin.html
- sudo /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python -m easy_install neovim
- let g:python_host_prog = '/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python'
- Don't do the following - instead do the above:
- brew install python2
- if python3 installed, you may need to run: brew link --overwrite python@2
- pip install neovim

# Add DPI scaling

Header Text

- Two levels of abstraction
  - Graphics level is dealing with pixels at a one to one pixel level - no scaling, just raw pixels
  - Painter level (and Widget level) which deals in logical units - eg pixels with some adjustable scale factor
  - Like text point vs pixel sizes - text at painter/widget level is in points, at graphis level it is pixels
- Currently just useRetina on/off, and this is a fixed scale of 2, but idea is same just an adjustable scale factor
- This might imply an API at one layer in floats, and then in other layer in ints/fixed point

# Add introspection

- how about query of .so files to find symbols and figure out stuff from that - demangle etc
- add list-views, tree views to navigate objects
- object trees
  - widgets inherit from object

# In Retina Mode

- Text in font-test doesn't clip to edge of screen
- The progress bar doesn't display properly
- The check-mark and scroll-bar handlers aren't resized / scaled / higher res
- Resize to new width/height, not sure it handles values for retina properly
- resizing window crashes

# In Normal or both Modes

- Various widgets aren't resized correctly to the required size for the contents / text
- The whole window dimensions aren't being used to resize the first child widget
- Combo-box menu list item highlight is wrong color

# Other

- Make it a runtime thing to switch to retina-mode as probably won't have diff builds for this
- Perhaps make it adjustable as to how much the scale factor is

# Idea

- font anti-aliasing
    - small fonts do anti-aliasing as currently done (fast to generate and cache)
    - for similar sized fonts, anti-alias off a single larger cached glyph
        - idea is the larger one can say be between 2 to 3 times the size
        - so if have a 10pt font, anti-alias off a 30pt font
            - 12pt font off the 30pt font
            - 15pt font off the 30pt font
            - 16pt font would then be off a 48pt font

    4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 … 35 36 37 38 … 57 58
    10 10 18 18 18 18 30 30 30 30 30 30 48 48 48 48 48 48 48 48 48 75 75 75 … 75 75 75 114 … 114 174

    4 … 58 (54 glyphs to create and cache)
    10, 18, 30, 48, 75, 114, 174 … ( 7 glyphs to create and cache)

    - for smaller fonts, it makes sense to anti-alias by rendering to a larger size and then down-sample the entire glyph
        - this is because of fine details in the glyph
    - but for larger sizes, the anti-aliasing isn't needed for helping capture the fine-details in a small size, but to give the outline a smooth appearance
    - so for larger sizes, another strategy instead of anti-alias by down-sampling, is to instead render the
      filled inside of the glyph without anti-aliasing, and then using an anti-aliasing line drawing algorithm to
      raw the smooth outline over the top of the filled inside (order may need to be reversed - fill drawn after)
    - sub-pixel rendering - an improvement over plain anti-aliasing is a technique to take in to account the RGB segments in a LCD panel which can give additional horizontal positioning accuracy by 3 times by using different colors on the edges. A left edge might be redish and a right edge bluish to allow them to extend 1/3 of a pixel over. (assuming black text on white. if white text on black, the colors are reversed, left is blueish, right is redish) It does require knowledge of the LCD properties - take a phone, rotate the screen, the LCD segments didn't rotate, so you need to know the orientation of what you are displaying to the physical LCD segments it is displayed on

(assuming even an LCD).

- Modifying the color has some implications for when you blend or combine over a background, as the outline with coloring
  may become obvious, particularly for a larger sized font.