**Optimizers and Tasks**

List of optimizers uses instead of `synfig::optimize_layers`. So each aspect of optimization procedure become more clear. Each `Renderer` class contains own list of optimizers, so we can configure optimization for any type of `renderer`. We can easy enable or disable any unstable optimizer dinamically.

Optimizers also uses to select type of rendering. For each common task exists a renderer-specified implementation and related optimizer-converter. For example, `OptimizerContourSW` converts common `TaskContour` into renderer-specified `TaskContourSW`. So `RendererSW` contains `OptimizerContourSW` in list of optimizers, and `RendererGL` contains `OptimizerContourGL`.

Also when `RendererGL` cannot draw something (task or optimizer-converter not implemented), we can associate software optimizer with it. And some tasks will drawn by software.

So synfig will automatically select hardware or software rendering.

**Example:**

Optimizers list associated with selected renderer:
- OptimizerSurfaceCreate (detect points where we need to initialize new surface)
- OptimizerSurfaceDestroy (detect points where we can remove surface)
- OptimizerLinear (reorganize task-tree to make linear list of tasks)
- OptimizerContourGL (draw contours by OpenGL)
- OptimizerGradientSW (draw gradients by Software)
- OptimizerBlendGL (blending by OpenGL)

Tasks before optimization
- TaskBlend
  - subtasks:
    - TaskContour
    - TaskGradient

Tasks after optimization:
- TaskSurfaceCreate (for contour)
- TaskContourGL
- TaskSurfaceCreate (for gradient)
- TaskGradientSW
- TaskSurfaceCreate (for blend)
- TaskBlendGL
- TaskSurfaceDestroy (for contour)
- TaskSurfaceDestroy (for gradien)

Gradient will rendered by software and will automatically converted into OpenGL texture which will be passed into `BlendGL`. Surface from `BlengGL` will not removed, because it stores rendering result.

Today we have only minimal set of optimizers to make valid task-list for software renderer. So our goal for future is to write optimizers for increase speed.
How rendering works today

// Build task
Context::build_rendering_task() {
    ... 
    Layer::build_rendering_task() {
        ...
        Context::build_rendering_task()
            ...
    }  
    ... 
}

// Create and assign target surface to store result of rendering
...

// Create task-list and insert single task into it
...

// Pass task-list to selected renderer
rendering::Renderer::run(task_list) {
    // Build optimized task-list
    rendering::Renderer::optimize(task_list) {
        foreach(optimizer in registered_optimizer)
            foreach(task in task_list)
                rendering::Optimizer::run()
    }

    // Render
    foreach(task in optimized_task_list)
        rendering::Task::run()
}
Our goal

// Build task-list
foreach(frame in scheduled_frames) {
    Context::build_rendering_task() {
        ...
        Layer:: build_rendering_task() {
            ...
            Context::build_rendering_task()
        }
        ...
    }
    ...
    // Create and assign target surface to store result
    ...
    // Insert task into task-list
    ...
}

// Pass task-list to selected renderer
rendering::Renderer::run(task_list) {
    // Build optimized task-list and mark dependencies
    rendering::Renderer::optimize(task_list) {
        foreach(optimizer in registered_optimizer)
            foreach(task in task_list)
                rendering::Optimizer::run()
    }

    // Multithreaded rendering
    while(!optimized_task_list.empty()) {
        // Wait for any unbussy thread
        ...
        // Find (or wait) task with complete dependecies
        ...
        rendering::Renderer::Thread::enqueue(task)
    }
}