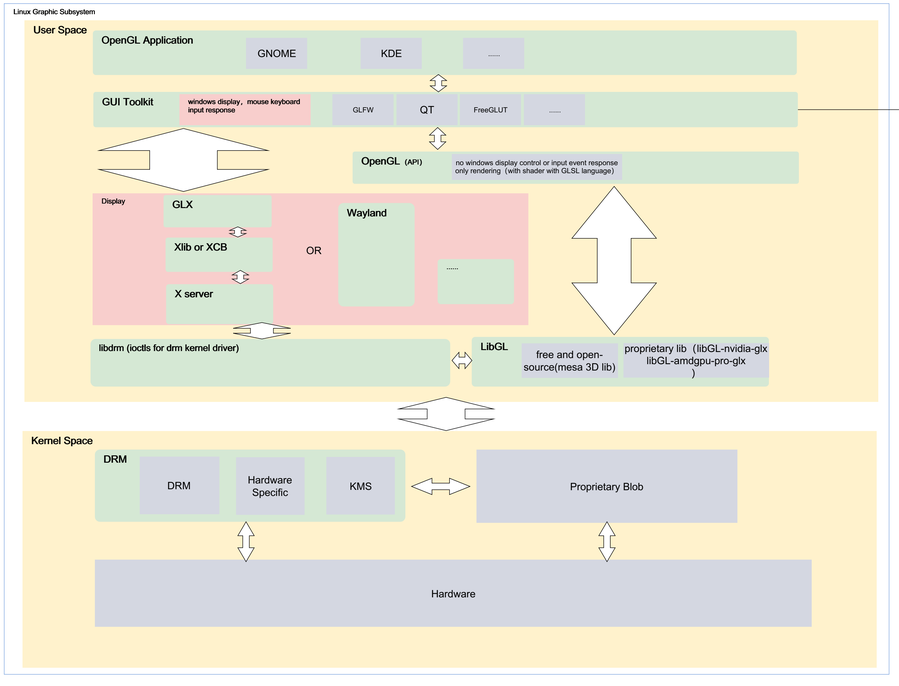
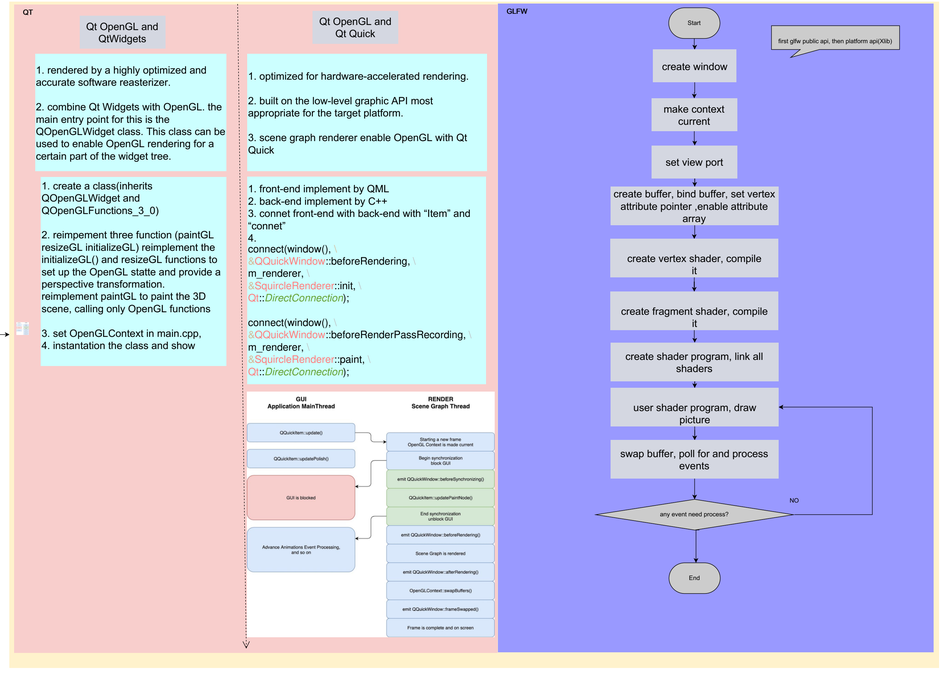

Linux Graphic Subsystem
User Space
OpenGL Application
GNOME
KDE
.....
GUI Toolkit
windows display，mouse keyboard input response
GLFW
QT
FreeGLUT
.....
OpenGL (API)
no windows display control or input event response only rendering（with shader with GLSL language）
Display
GLX
Xlib or XCB
X server
OR
Wayland
.....
libdrm (ioctls for drm kernel driver)
LibGL
free and open-source(mesa 3D lib)
proprietary lib（libGL-nvidia-glx libGL-amdgpu-pro-glx ）
Kernel Space
DRM
DRM
Hardware Specific
KMS
Proprietary Blob
Hardware


QT
Qt OpenGL and QtWidgets
1. rendered by a highly optimized and accurate software reasterizer.
2. combine Qt Widgets with OpenGL. the main entry point for this is the QOpenGLWidget class. This class can be used to enable OpenGL rendering for a certain part of the widget tree.
1. create a class(inherits QOpenGLWidget and QOpenGLFunctions_3_0)
2. reimpement three function (paintGL resizeGL initializeGL) reimplement the initializeGL() and resizeGL functions to set up the OpenGL statte and provide a perspective transformation. reimplement paintGL to paint the 3D scene, calling only OpenGL functions
3. set OpenGLContext in main.cpp,
4. instantation the class and show
Qt OpenGL and Qt Quick
1. optimized for hardware-accelerated rendering.
2. built on the low-level graphic API most appropriate for the target platform.
3. scene graph renderer enable OpenGL with Qt Quick
1. front-end implement by QML
2. back-end implement by C++
3. connet front-end with back-end with “Item” and “connet”
4.
connect(window(), \
&QQuickWindow::beforeRendering, \
m_renderer, \
&SquircleRenderer::init, \
Qt::DirectConnection);
connect(window(), \
&QQuickWindow::beforeRenderPassRecording, \
m_renderer, \
&SquircleRenderer::paint, \
Qt::DirectConnection);
GUI
Application MainThread
RENDER
Scene Graph Thread
QQuickItem::update()
QQuickItem::updatePolish()
GUI is blocked
Advance Animations Event Processing, and so on
Starting a new frame OpenGL Context is made current
Begin synchronization block GUI
emit QQuickWindow::beforeSynchronizing()
QQuickItem::updatePaintNode()
End synchronization unblock GUI
emit QQuickWindow::beforeRendering()
Scene Graph is rendered
emit QQuickWindow::afterRendering()
OpenGLContext::swapBuffers()
emit QQuickWindow::frameSwapped()
Frame is complete and on screen
GLFW
Start
first glfw public api, then platform api(Xlib)
create window
make context current
set view port
create buffer, bind buffer, set vertex attribute pointer ,enable attribute array
create vertex shader, compile it
create fragment shader, compile it
create shader program, link all shaders
user shader program, draw picture
swap buffer, poll for and process events
any event need process?
NO
End