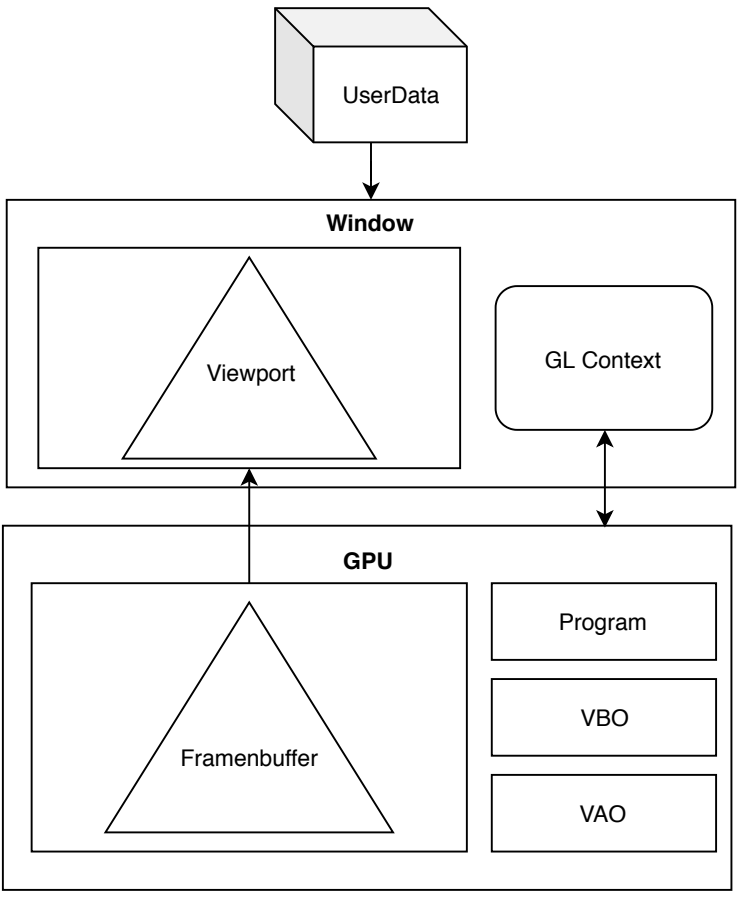
main.c

- GLFW initialisieren
- Fenster erstellen
- UserData allokieren
- Grafik-Kontext erstellen
- (endlos-)Render-Schleife starten
- (Callback für Nutzerinteraktion registrieren)



glCalls.c

init()
- (Vertex/Fragment)Shader laden und kompilieren
  - leeres Shaderobjekt erstellen
  - Shadercode dem Shaderobjekt zuweisen
  - Kompilieren
  - Fehlerbehandlung
  - Shader Compiler freigeben
- Vollständiges Shader-Progamm erstellen (= Vertex+Fragmentshader linken)
  - leeres Programmobjekt erstellen
  - beide Shader dem "Programm" zuweisen
  - Shader linken -> Vollständiges Programm
  - Fehlerbehandlung
  - Programm verwenden

- Tatsächliche Dreiecksdaten anlegen

- VAO erstellen und binden

- Buffer (VBO) für Dreiecksdaten auf der Grafikkarte anlegen
  - leeres VertexBufferObject erstellen
  - "Referenz" auf VBO in UserData speichern
  - Dreiecksdaten in das VBO laden (RAM -> Grafikkarten-Speicher)
  - Der Grafikkarte mitteilen, was im VBO steht und wie das VBO zu lesen ist -> VAO

- Framebuffergröße aka Fenstergröße ermitteln
- glViewport setzen -> Größe der GL-Leinwand (Framebuffer) OpenGL mitteilen
- (Hinter)Grundfarbe des Framebuffers definieren

draw()
- Framebuffer mit Grundfarbe "übermalen"
- Zeichenmethode aufrufen (Woher die Daten? Was wird gezeichnet? Verwirrung? -> OpenGL = state machine!)
- Framebuffer Swap aka Framebuffer ins Fenster schreiben

teardown()
- auf der Grafikkarte aufräumen
- VBO löschen
- VAO löschen
- Program löschen
- Allokierte Nutzerdaten (UserData) löschen/freigeben