

Single 3D frame

0) Initialize libraries, register callbacks, ...

1) Set initial projection, lights, camera, ...

1) Recompute app state values (health, positions, ...)

2) Frame start → Clear canvas

3) Draw scene → Vertices with attributes

4) Finish frame → Display

5) Poll OS events,

2) End program



GLFW Functionality Overview

(third party library used in labs)

- library for OpenGL, OpenGL ES, Vulkan apps
- multiplatform, multilanguage
- Z-Buffer, stencil, alpha settings
- Camera settings
- Callbacks for OS events
 - mouse, keyboard, timer, windows resize, window redraw, ...
- NO fonts, GUI, sound, texture load, threads, ...

GLFW

Hello world

```
#include <GLFW/glfw3.h>

int main(void)
{
    GLFWwindow* window;

    /* Initialize the library */
    if (!glfwInit())
        return -1;

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(640, 480, "Hello World", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }

    /* Make the window's context current */
    glfwMakeContextCurrent(window);

    /* Loop until the user closes the window */
    while (!glfwWindowShouldClose(window))
    {
        /* Render here */
        glClear(GL_COLOR_BUFFER_BIT);

        /* ... more draw-calls ... */

        /* Swap front and back buffers */
        glfwSwapBuffers(window);

        /* Poll for and process events */
        glfwPollEvents();
    }

    glfwTerminate();
    return 0;
}
```

GLFW Init, callback

glfwSetErrorCallback – error can occur during initialization
glfwInit

```
//=====
//callback definition for GLFW
void error_callback(int error, const char* description)
{
    std::cerr << "Error: " << description << std::endl;
}

//=====

void init_glfw(void)
{
    // set error callback first
    glfwSetErrorCallback(error_callback);

    //initialize GLFW library
    int glfw_ret = glfwInit();
    if (!glfw_ret) {
        std::cerr << "GLFW init failed." << std::endl;
        exit(EXIT_FAILURE);
    }

    window = glfwCreateWindow(800, 600, "OpenGL context", NULL, NULL);
    if (!window) {
        std::cerr << "GLFW window creation error." << std::endl;
        exit(EXIT_FAILURE);
    }

    // Set current window.
    glfwMakeContextCurrent(window);
}
```

Querying basic info

glfwGetVersion
glfwGetVersionString

```
// Get some GLFW info.
{
    int major, minor, revision;

    glfwGetVersion(&major, &minor, &revision);
    std::cout << "Running GLFW " << major << '.' << minor << '.' << revision << '\n';
    std::cout << "Compiled against GLFW "
        << GLFW_VERSION_MAJOR << '.' << GLFW_VERSION_MINOR << '.' << GLFW_VERSION_REVISION
        << '\n';
}
```

END of application loop with GLFW

```
void window_close_callback(GLFWwindow* window)
{
    if (!time_to_close)
        glfwSetWindowShouldClose(window, GLFW_FALSE); // You can cancel the request.
}

int main(void)
{
    glfwSetWindowCloseCallback(window, window_close_callback);

    while (!glfwWindowShouldClose(window)) // APP loop
    {
        recompute_physics_etc();

        do_my_render(window);

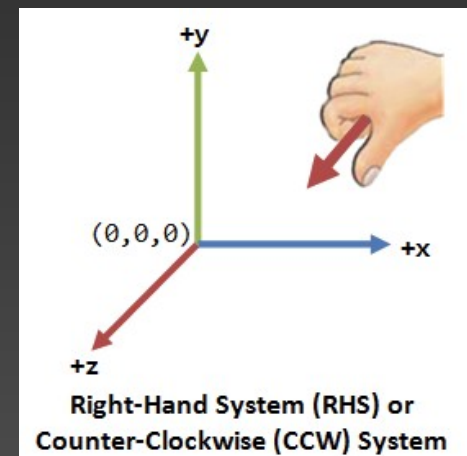
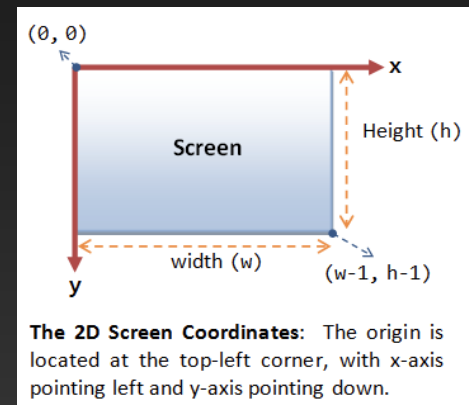
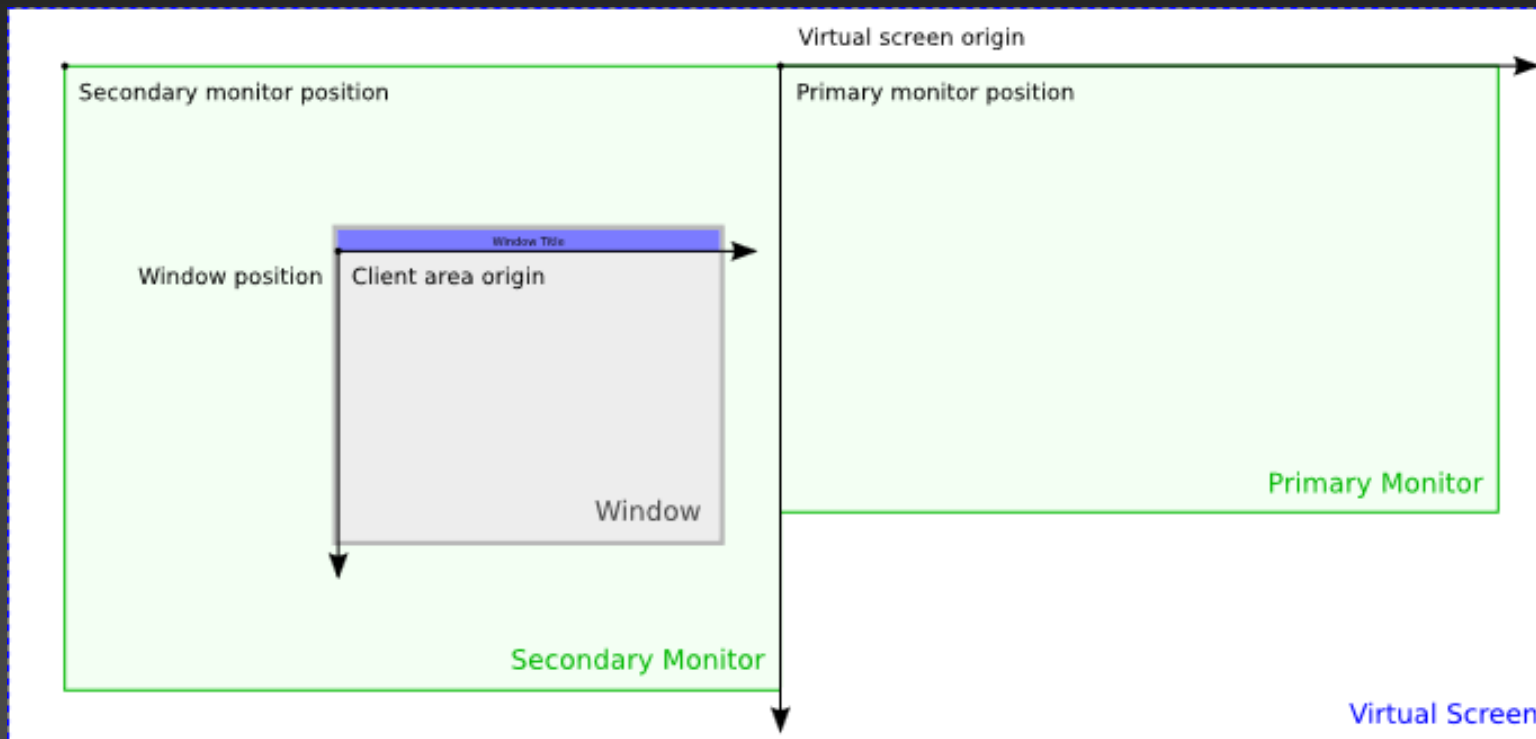
        if (i_want_to_quit)
            glfwSetWindowShouldClose(window, GLFW_TRUE);

        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // Close OpenGL window if opened and terminate GLFW
    if (window)
        glfwDestroyWindow(window);
}
```

GLFW coordinates

- Virtual screen coordinates and client area
 - **may** map 1:1 to pixels
 - eg. not on Mac with a Retina display etc.



GLFW and threads

- All callbacks and event processing must be called in **main thread**!
- **Rendering** may be done on **any single thread**.
- Event processing and obj. destruction are **NOT** reentrant → **must not** call these in callbacks:

glfwDestroyWindow
glfwDestroyCursor
glfwPollEvents
glfwWaitEvents
glfwWaitEventsTimeout
glfwTerminate

GLFW window management

```
// Create in windowed mode
GLFWwindow* window = glfwCreateWindow(640, 480, "My Title", NULL, NULL);

// Create in fullscreen mode
GLFWwindow* window = glfwCreateWindow(640, 480, "My Title", glfwGetPrimaryMonitor(), NULL);

// Switch windowed and fullscreen
// Get primary monitor
GLFWmonitor* monitor = glfwGetPrimaryMonitor();
// Get resolution of monitor
const GLFWvidmode* mode = glfwGetVideoMode(monitor);
// Switch to full screen
glfwSetWindowMonitor(window, monitor, 0, 0, mode->width, mode->height, mode->refreshRate);

// Switch back to windowed mode
glfwSetWindowMonitor(window, nullptr, xpos, ypos, width, height, 0);

void window_iconify_callback(GLFWwindow* window, int iconified)
{
    if (iconified) {
        // The window was iconified
    } else {
        // The window was restored
    }
}

// minimize
glfwIconifyWindow(window);

// restore
glfwRestoreWindow(window);

// set callback
glfwSetWindowIconifyCallback(window, window_iconify_callback);
```

GLFW window size limits

- Get

```
glfwGetWindowSize(window, &width, &height);
```

- Set minimum & maximum

```
glfwSetWindowSizeLimits(window, 200, 200, 400, 400);
```

- Set only minimum

```
glfwSetWindowSizeLimits(window, 640, 480, GLFW_DONT_CARE, GLFW_DONT_CARE);
```

- Force aspect ratio

```
glfwSetWindowAspectRatio(window, 16, 9);
```

GLFW window size

- Do **not** pass the **window** size to `glViewport` or other pixel-based OpenGL calls. The window size is in **screen** coordinates, not pixels. Use the **framebuffer** size, which is in **pixels**, for pixel-based calls.

```
void window_size_callback(GLFWwindow* window, int width, int height)
{
}

void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}

main ()
{
    int width, height;
    glfwSetWindowSizeCallback(window, window_size_callback);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    glfwGetWindowSize(window, &width, &height);
    glfwSetWindowSize(window, 640, 480);

    glfwGetFramebufferSize(window, &width, &height);
}
```

GLFW – other functions

- set window focus

```
glfwFocusWindow(window);
```

- Window title, window icon, position, show & hide, scaling...
- Multi-monitor: positions, resolution, scale, area...

```
void window_focus_callback(GLFWwindow* window, int focused)
{
    if (focused) {
        // The window gained input focus
    } else {
        // The window lost input focus
    }
}

main ()
{
    glfwSetWindowFocusCallback(window, window_focus_callback);

    int focused = glfwGetWindowAttrib(window, GLFW_FOCUSED);
}
```

GLFW event processing

- **continuous** (games)

`glfwPollEvents();`

- only on input (force thread to sleep)

`glfwWaitEvents();`

- on input with timeout in seconds

`glfwWaitEventsTimeout(0.7);`

- wake main thread sleeping on `glfwWaitEvents()` from another thread

`glfwPostEmptyEvent();`

- Swaps the front and back buffers of the window

`glfwSwapBuffers` (window)

GLFW keyboard input

- key codes – defined for portability
 - GLFW_KEY_0 .. GLFW_KEY_9, GLFW_KEY_A .. GLFW_KEY_Z
 - GLFW_KEY_APOSTROPHE, GLFW_KEY_COMMA, GLFW_KEY_MINUS, ...
- standard key action
 - GLFW_PRESS, GLFW_REPEAT, GLFW_RELEASE
- unknown key action (multimedia keyboard – play, email, ...)
GLFW_KEY_UNKNOWN
- modifiers
GLFW_MOD_ALT, GLFW_MOD_CONTROL, GLFW_MOD_SHIFT, GLFW_MOD_SUPER
- Also sticky keys

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_E && action == GLFW_PRESS)
        activate_airship();
}

main ()
{
    glfwSetKeyCallback(window, key_callback);

    // or poll state manually
    int state = glfwGetKey(window, GLFW_KEY_E);
    if (state == GLFW_PRESS)
        activate_airship();
}
```

GLFW keyboard input - UNICODE

- supports composing characters using dead keys

```
void character_callback(GLFWwindow* window, unsigned int codepoint) { ... }
```

```
glfwSetCharCallback(window, character_callback);
```

- translate default key position to language layout

```
const char* key_name = glfwGetKeyName(GLFW_KEY_W, 0);
```

```
show_tutorial_hint("Press %s to move forward", key_name);
```

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_E && action == GLFW_PRESS)
        activate_airship();
}

main ()
{
    glfwSetKeyCallback(window, key_callback);

    // or poll state manually
    int state = glfwGetKey(window, GLFW_KEY_E);
    if (state == GLFW_PRESS)
        activate_airship();
}
```

GLFW mouse input

- measured in screen coordinates
 - relative to the top-left corner of the window client area
- actions: GLFW_PRESS, GLFW_RELEASE
- buttons: GLFW_MOUSE_BUTTON_LEFT (1), GLFW_MOUSE_BUTTON_RIGHT (2), GLFW_MOUSE_BUTTON_MIDDLE (3), GLFW_MOUSE_BUTTON_1 (4), GLFW_MOUSE_BUTTON_2 (5), GLFW_MOUSE_BUTTON_3 (6), GLFW_MOUSE_BUTTON_4 (7), GLFW_MOUSE_BUTTON_5 (8)
- Also sticky mouse buttons

```
static void cursor_position_callback(GLFWwindow* window, double xpos, double ypos)
{
}
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS)
        action();
}
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
}
main ()
{
    glfwSetCursorPosCallback(window, cursor_pos_callback);
    glfwSetMouseButtonCallback(window, mouse_button_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // or by polling
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);
    // or by polling
    int state = glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_LEFT);
    if (state == GLFW_PRESS)
        upgrade_cow();
}
```


GLFW cursor

- mouse motion based camera controls (fullscreen games)
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
- Raw mouse input: no acceleration etc.
if (glfwRawMouseMotionSupported())
glfwSetInputMode(window, GLFW_RAW_MOUSE_MOTION, GLFW_TRUE);
- other modes
GLFW_CURSOR_NORMAL, GLFW_CURSOR_HIDDEN
- Custom cursor possible

```
void cursor_enter_callback(GLFWwindow* window, int entered)
{
    if (entered) {
        // The cursor entered the client area of the window
    } else {
        // The cursor left the client area of the window
    }
}

main ()
{
    glfwSetCursorEnterCallback(window, cursor_enter_callback);

    // change cursor shape
    //GLFW_ARROW_CURSOR, GLFW_CROSSHAIR_CURSOR, GLFW_HAND_CURSOR, GLFW_HRESIZE_CURSOR
    //GLFW_VRESIZE_CURSOR, GLFW_IBEAM_CURSOR
    GLFWcursor* cursor = glfwCreateStandardCursor(GLFW_HRESIZE_CURSOR);

    // set shape
    glfwSetCursor(window, cursor);
    // no cursor
    glfwSetCursor(window, NULL);

    glfwDestroyCursor(cursor);
}
```

GLFW – other functionality

- Time from glfwInit(...) call

```
double seconds = glfwGetTime();
```

- Joystick

```
int = glfwJoystickPresent(GLFW_JOYSTICK_1)
char * = glfwGetJoystickName(GLFW_JOYSTICK_1)
uchar * = glfwGetJoystickButtons(GLFW_JOYSTICK_1, &count)
uchar * = glfwGetJoystickAxes(GLFW_JOYSTICK_1, &count)
uchar * = glfwGetJoystickHats(GLFW_JOYSTICK_1, &count);
```

- Gamepad

```
if (glfwJoystickIsGamepad(GLFW_JOYSTICK_2)) { /* Use as gamepad */ }
```

- Clipboard

```
glfwGetClipboardString, glfwSetClipboardString
```

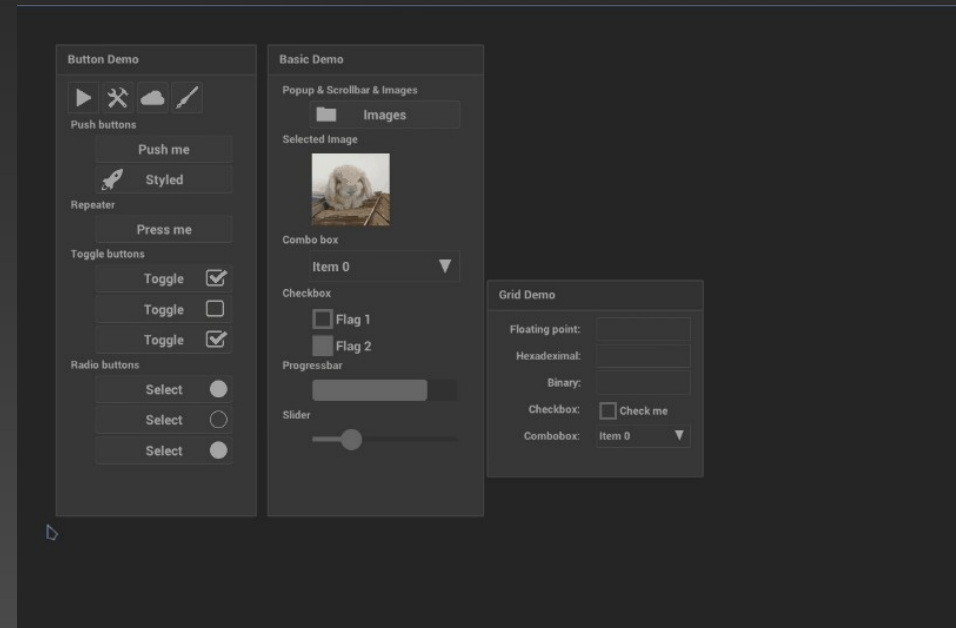
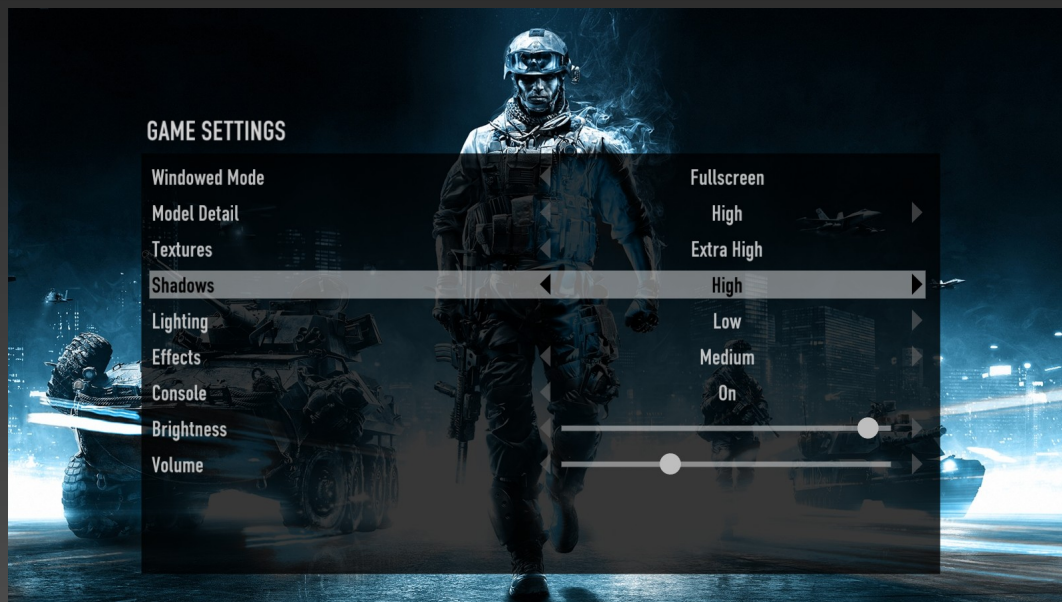
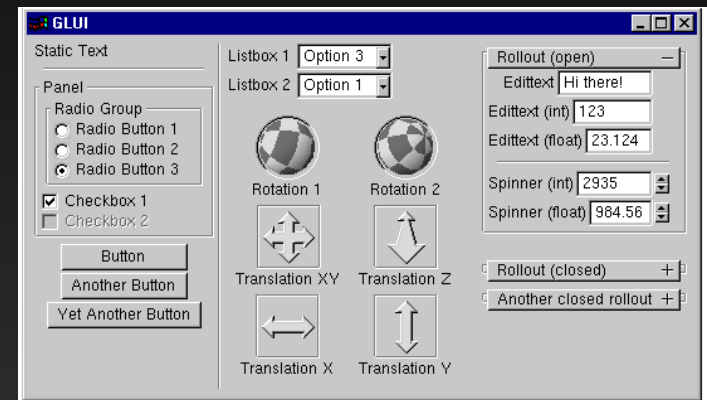
- File object drop

```
glfwSetDropCallback
```

- Vulkan support, platform dependent calls

How about GUI?

- GLFW library allows simple I/O and setting
 - no GUI, text, etc.
- More libraries needed!
 - GLUI (github)
 - Nuklear (github)



Single 3D frame draw

Single 3D frame

0) Initialize libraries, register callbacks, ...

1) Set initial projection, lights, camera, ...

1) Recompute app state values (health, positions, ...)

2) Frame start → Clear canvas

3) Draw scene → Vertices with attributes

4) Finish frame → Display

5) Poll OS events

2) End program



Drawing Single Frame – Start

- At start – clear canvas

```
glClearColor(RGBA)
```

```
glClear( BIT_MASK ) //what to clear
```

- Bitmask tells, what will be cleared

1) frame buffer (color buffer)

2) Z-buffer (depth buffer)

3) accumulation buffer

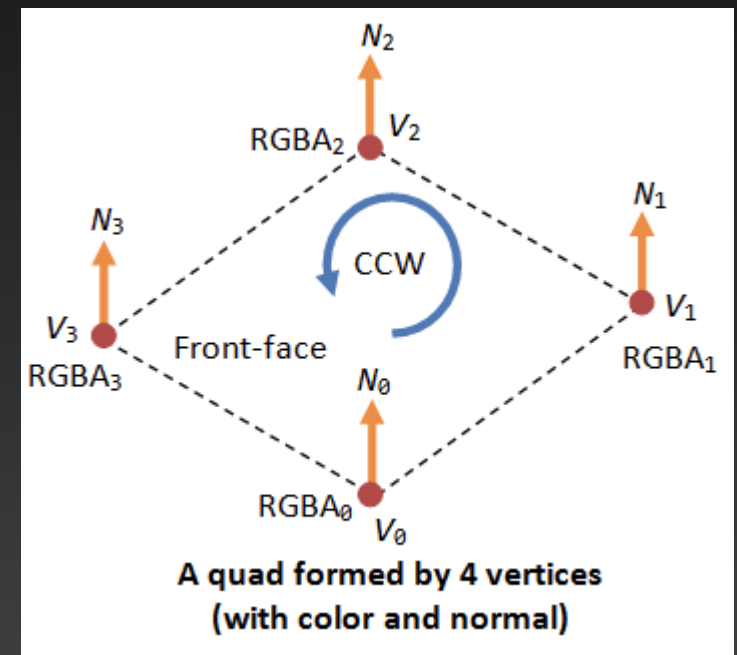
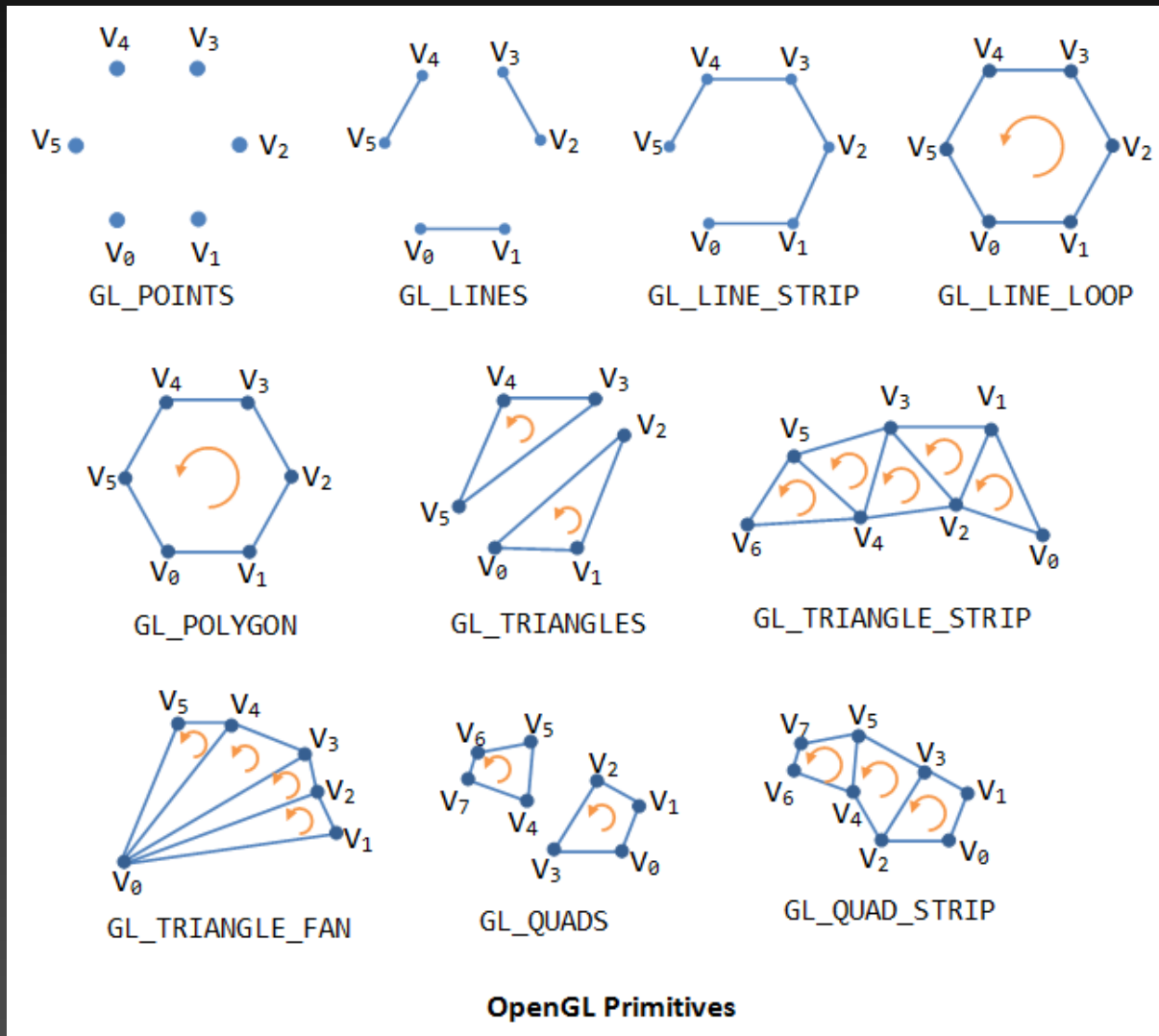
4) stencil buffer

```
glClearColor(0.0, 0.0, 0.0, 0.0);
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

OpenGL Graphical Primitives

- Use primitives to draw the whole scene



Drawing Single Frame – End

- All graphics primitives sent to GPU
- GO!
 - glFlush()
 - flush all buffers with primitives in driver etc., start draw immediately, **do not wait** till finish!
 - glFinish()
 - flush all buffers with primitives in driver etc., start draw immediately, do WAIT till finish!
 - whole application (thread) **blocked**

OpenGL Conventions

- Constants
 - Prefix **GL_** and all capitals (eg. GL_TRUE)
- Functions
 - Prefix **gl** and CamelCase (eg. glClearColor)
 - Variations of parameters
 - **glCommand**{-,2,3,4}{**b,s,i,f,d**,...}{-,v}(x1, y1, x2, y2, ...)
- Data types
 - 1) GL**byte**
 - 2) GL**short**
 - 3) GL**int**
 - 4) GL**float**
 - 5) GL**double**
 - ...