OpenGL

Base engine overview

Advanced Graphics Programming

Platform

OpenGL version (platform.cpp)

```
if (!glfwInit())
{
    ELOG("glfwInit() failed\n");
    return -1;
}

glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

We will use **OpenGL 4.3 Core profile**.

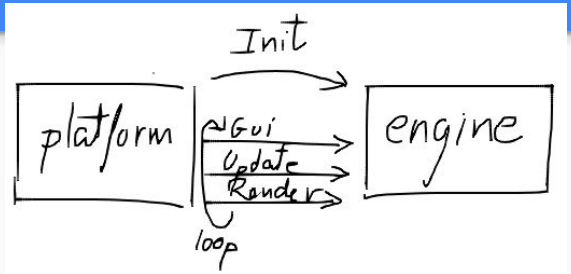
We don't use the compatibility profile to avoid calling deprecated OpenGL functions.

There are newer OpenGL versions with more and more function calls. You can increase the version it if at some you require some newer functionality.

However, all the contents given in this subject can be implemented using 4.3 with no problem.

Application life cycle (platform.cpp)

```
Init(&app);

while (app.isRunning)
{
    glfwPollEvents();

    // ImGui frame initialization...

    Gui(&app);

    // More ImGui handling stuff...

    // Update
    Update(&app);

    // Input event transitions...

    Render(&app);

    // ImGui overlay render...

    glfwSwapBuffers(window);

    // Frame time...
}
```



We implement these functions in engine.cpp.

These are the typical Application class functions. Just another way of doing it.

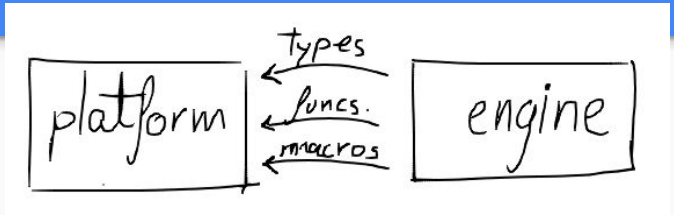
Feel free to modularize the code if you feel more comfortable… but we will not be implementing more modules than the render module.

What are they for?

- Init: to initialize one-time-load resources
- Gui: to program our imgui controls
- Update: to implement shortcuts and camera movement
- Render: real-time opengl rendering code

Platform interface

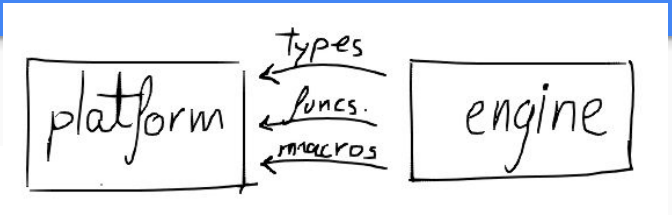
Platform utils (platform.h)



Sized types

```
typedef char                 i8;
typedef short                i16;
typedef int                  i32;
typedef long long int        i64;
typedef unsigned char        u8;
typedef unsigned short       u16;
typedef unsigned int         u32;
typedef unsigned long long int u64;
typedef float                f32;
typedef double               f64;
```

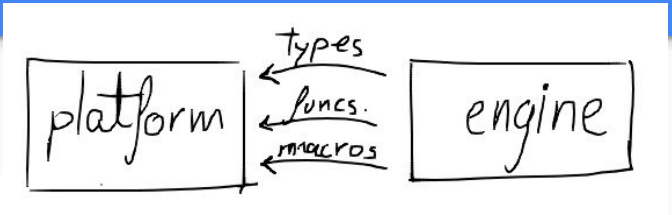
Platform utils (platform.h)



Input types

```
enum MouseButton {
    LEFT,
    RIGHT,
    MOUSE_BUTTON_COUNT
};

enum Key {
    K_SPACE,
    K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9,
    K_A, K_B, K_C, K_D, K_E, K_F, K_G, K_H, K_I, K_J, K_K, K_L, K_M,
    K_N, K_O, K_P, K_Q, K_R, K_S, K_T, K_U, K_V, K_W, K_X, K_Y, K_Z,
    K_ENTER, K_ESCAPE,
    KEY_COUNT
};

enum ButtonState {
    BUTTON_IDLE,
    BUTTON_PRESS,
    BUTTON_PRESSED,
    BUTTON_RELEASE
};

struct Input {
    glm::vec2   mousePos;
    glm::vec2   mouseDelta;
    ButtonState mouseButtons[MOUSE_BUTTON_COUNT];
    ButtonState keys[KEY_COUNT];
};
```

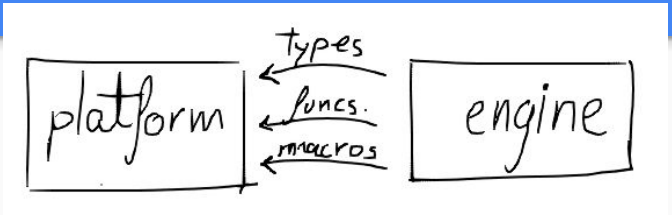
Platform utils (platform.h)



Strings / File functions

```
struct String
{
    char* str;
    u32   len;
};

String MakeString(const char *cstr);

String MakePath(String dir, String filename);

String GetDirectoryPart(String path);

/* ... */
String ReadTextFile(const char *filepath);

/* ... */
u64 GetFileLastWriteTimestamp(const char *filepath);
```

Platform utils (platform.h)



Macros

```
#define ILOG(...)                    \
{ ... }

#define ELOG(...) ILOG(__VA_ARGS__)

#define ARRAY_COUNT(array) (sizeof(array)/sizeof(array[0]))

#define ASSERT(condition, message) assert((condition) && message)

#define KB(count) (1024*(count))
#define MB(count) (1024*KB(count))
#define GB(count) (1024*MB(count))

#define PI  3.14159265359f
#define TAU 6.28318530718f
```

Engine types

Engine types (engine.h)

```
struct Texture
{
    GLuint      handle;
    std::string filepath;
};

struct Program
{
    GLuint              handle;
    std::string         filepath;
    std::string         programName;
    u64                 lastWriteTimestamp; // What is this for?
};
```

Engine types (engine.h)

```
struct Texture
{
    GLuint      handle;
    std::string filepath;
};

struct Program
{
    GLuint              handle;
    std::string         filepath;
    std::string         programName;
    u64                 lastWriteTimestamp; // What is this for?
};
```

Engine types (engine.h)

```
struct App
{
    // Loop
    f32  deltaTime;
    bool isRunning;

    // Input
    Input input;

    // Graphics
    char gpuName[64];
    char openGlVersion[64];

    ivec2 displaySize;

    std::vector<Texture>  textures;
    std::vector<Program>  programs;

    // program indices
    u32 texturedGeometryProgramIdx;

    // texture indices
    u32 diceTexIdx;
    u32 whiteTexIdx;
    u32 blackTexIdx;
    u32 normalTexIdx;
    u32 magentaTexIdx;

    // Mode
    Mode mode;

    // Embedded geometry (in-editor simple meshes such as
    // a screen filling quad, a cube, a sphere...)
    GLuint embeddedVertices;
    GLuint embeddedElements;

    // Location of the texture uniform in the textured quad shader
    GLuint programUniformTexture;

    // VAO object to link our screen filling quad with our textured quad shader
    GLuint vao;
};
```

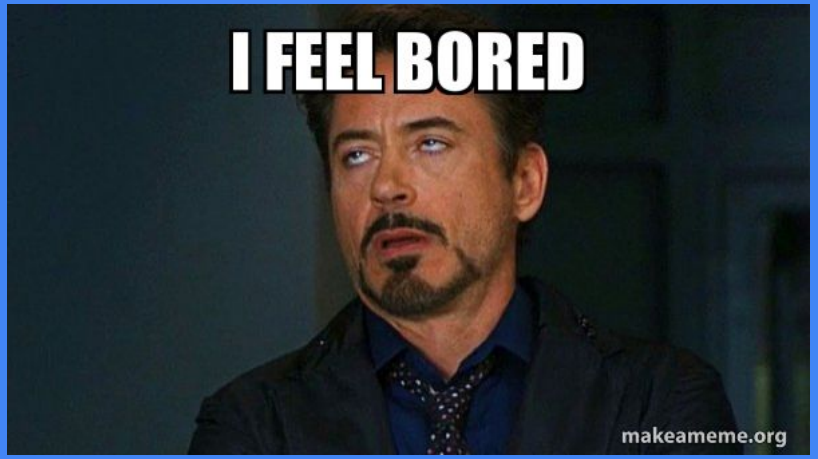
TODO:
Retrieve OpenGL information

TODO: Retrieve OpenGL information

With Dear ImGui, create a new window showing information about the OpenGL context.

- You can retrieve the information only once in the **initialization** function.
- Instead of printing this information as in the image, store it into some OpenGLInfo struct within our application and show it on a dialog when a certain button or menu action is pressed.

```
std::cout << "OpenGL version:" << std::endl;
std::cout << glGetString(GL_VERSION) << std::endl << std::endl;

std::cout << "OpenGL renderer:" << std::endl;
std::cout << glGetString(GL_RENDERER) << std::endl << std::endl;

std::cout << "OpenGL vendor:" << std::endl;
std::cout << glGetString(GL_VENDOR) << std::endl << std::endl;

std::cout << "OpenGL GLSL version:" << std::endl << std::endl;
std::cout << glGetString(GL_SHADING_LANGUAGE_VERSION) << std::endl;

std::cout << "OpenGL extensions:" << std::endl;
GLint num_extensions;
glGetIntegerv(GL_NUM_EXTENSIONS, &num_extensions);
for (int i = 0; i < num_extensions; ++i)
{
    const unsigned char *str = glGetStringi(GL_EXTENSIONS, GLuint(i));
    std::cout << str << " ";
}
std::cout << std::endl;
```

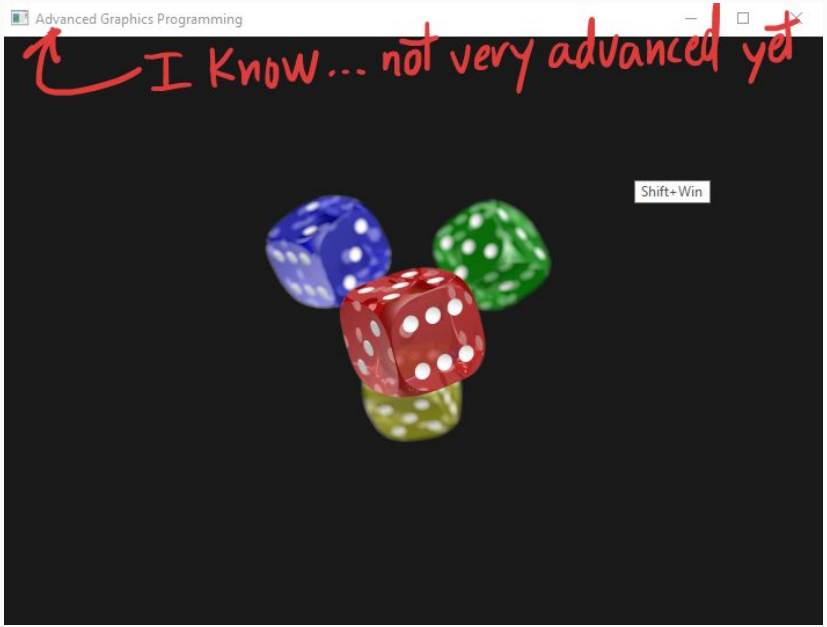


TODO:
Let’s paint a textured quad

TODO: Let's paint a textured quad

To paint this quad we will need to prepare some OpenGL objects:

- An **VBO** or **vertex buffer object** (an array in the GPU with the actual vertex data: 3D and texture coords)
- An **EBO** or **element buffer object** (an array in the GPU with the indices in the order they should be drawn)
- A **VAO** or **vertex array object** (an object that links the vertex buffer with the program)
- A **shader program** (the code that we will install in the GPU, that will position and paint the quad)
- A **texture**.



TODO: Let’s paint a textured quad (GLSL code)

```
3   ////////////////////////////////////////////////////////////////////
4   #ifdef TEXTURED_GEOMETRY
5
6   #if defined(VERTEX) ////////////////////////////////////////////////
7
8   layout(location=0) in vec3 aPosition;
9   layout(location=1) in vec2 aTexCoord;
10
11  out vec2 vTexCoord;
12
13  void main()
14  {
15      vTexCoord = aTexCoord;
16      gl_Position = vec4(aPosition, 1.0);
17  }
18
19  #elif defined(FRAGMENT) ////////////////////////////////////////////
20
21  in vec2 vTexCoord;
22
23  uniform sampler2D uTexture;
24
25  layout(location = 0) out vec4 oColor;
26
27  void main()
28  {
29      oColor = texture(uTexture, vTexCoord);
30  }
31
32  #endif
33  #endif
```

TODO: Let’s paint a textured quad (VBO and EBO initialization)

```
struct VertexV3V2
{
    glm::vec3 pos;
    glm::vec2 uv;
};

const VertexV3V2 vertices[] = {
    { glm::vec3(-0.5, -0.5, 0.0), glm::vec2(0.0, 0.0) }, // bottom-left vertex
    { glm::vec3( 0.5, -0.5, 0.0), glm::vec2(1.0, 0.0) }, // bottom-right vertex
    { glm::vec3( 0.5,  0.5, 0.0), glm::vec2(1.0, 1.0) }, // top-right vertex
    { glm::vec3(-0.5,  0.5, 0.0), glm::vec2(0.0, 1.0) }, // top-left vertex
};

const u16 indices[] = {
    0, 1, 2,
    0, 2, 3
};

// Geometry
glGenBuffers(1, &app->embeddedVertices);
glBindBuffer(GL_ARRAY_BUFFER, app->embeddedVertices);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glGenBuffers(1, &app->embeddedElements);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, app->embeddedElements);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Vertex shader (to have context)

```
///////////////////////////////////////
#ifdef TEXTURED_GEOMETRY

#if defined(VERTEX) ////////////////////

layout(location=0) in vec3 aPosition;
layout(location=1) in vec2 aTexCoord;

out vec2 vTexCoord;

void main()
{
    vTexCoord = aTexCoord;
    gl_Position = vec4(aPosition, 1.0);
}
```

TODO: Let's paint a textured quad (VAO initialization)

```
// Attribute state
glGenVertexArrays(1, &app->vao);
glBindVertexArray(app->vao);
glBindBuffer(GL_ARRAY_BUFFER, app->embeddedVertices);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexV3V2), (void*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(VertexV3V2), (void*)12);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, app->embeddedElements);
glBindVertexArray(0);
```

Vertex shader (to have context)

```
3   ////////////////////////////////////////
4   #ifdef TEXTURED_GEOMETRY
5
6   #if defined(VERTEX) ////////////////////
7
8   layout(location=0) in vec3 aPosition;
9   layout(location=1) in vec2 aTexCoord;
10
11  out vec2 vTexCoord;
12
13  void main()
14  {
15      vTexCoord = aTexCoord;
16      gl_Position = vec4(aPosition, 1.0);
17  }
```

Between glBindVertexArray(handle) and glBindVertexArray(0) VAOs record the state described by the function calls in the middle:

- Enabled vertex attributes
- Pointers into the buffers for those enabled vertex attributes
- Bound element array buffers (indices)

TODO: Let’s paint a textured quad (Program initialization)

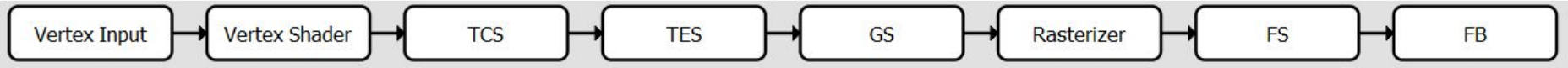
```
app->texturedGeometryProgramIdx = LoadProgram(app, "shaders.glsl", "TEXTURED_GEOMETRY");

Program& texturedGeometryProgram = app->programs[app->texturedGeometryProgramIdx];

app->programUniformTexture = glGetUniformLocation(texturedGeometryProgram.handle, "uTexture");
```

TODO: Let’s paint a textured quad (Texture initialization)

```
app->diceTexIdx = LoadTexture2D(app, "dice.png");
app->whiteTexIdx = LoadTexture2D(app, "color_white.png");
app->blackTexIdx = LoadTexture2D(app, "color_black.png");
app->normalTexIdx = LoadTexture2D(app, "color_normal.png");
app->magentaTexIdx = LoadTexture2D(app, "color_magenta.png");
```

Now really…
let’s paint a textured quad!

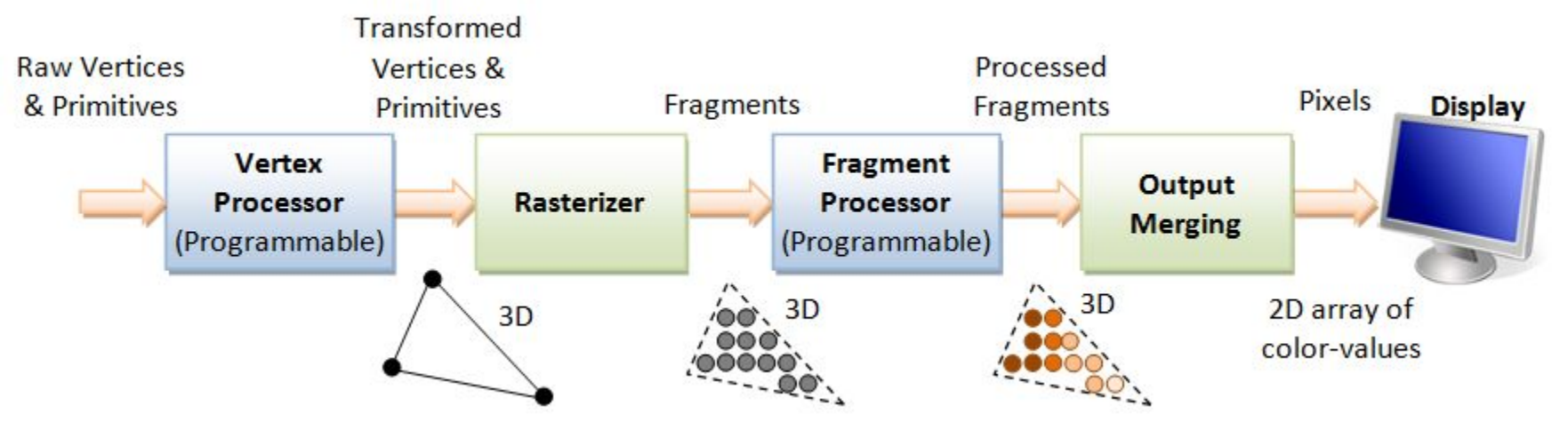
TODO: Let’s paint a textured quad

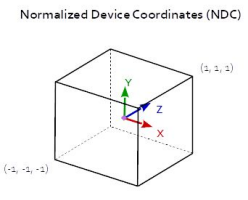


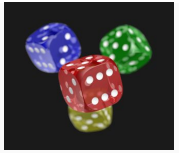

```
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glViewport(0, 0, app->displaySize.x, app->displaySize.y);

Program& programTexturedGeometry = app->programs[app->texturedGeometryProgramIdx];
glUseProgram(programTexturedGeometry.handle);
glBindVertexArray(app->vao);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glUniform1i(app->programUniformTexture, 0);
glActiveTexture(GL_TEXTURE0);
GLuint textureHandle = app->textures[app->diceTexIdx].handle;
glBindTexture(GL_TEXTURE_2D, textureHandle);

glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0);

glBindVertexArray(0);
glUseProgram(0);
```

TODO: Let's paint a textured quad

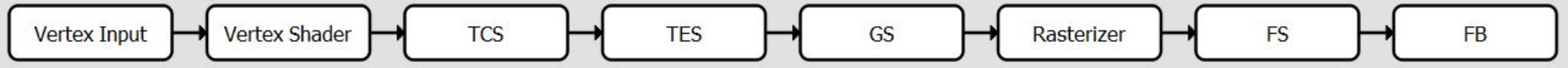
When we perform the rendering call, the following pipeline will execute in the GPU:





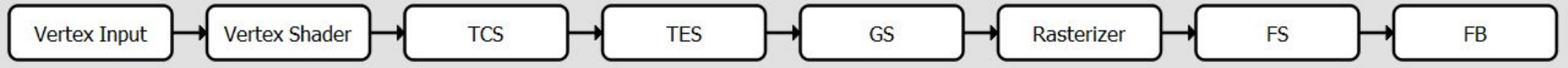


TODO: Let’s paint a textured quad



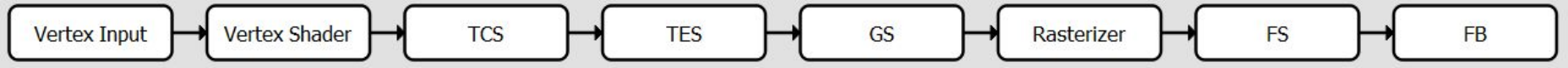

```
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

TODO: Let’s paint a textured quad



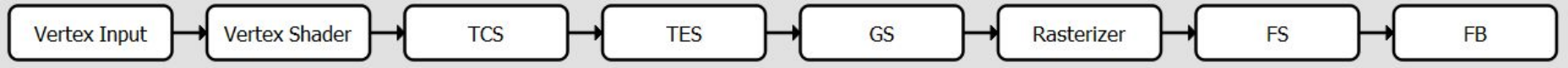

```
glViewport(0, 0, app->displaySize.x, app->displaySize.y);
```

TODO: Let’s paint a textured quad



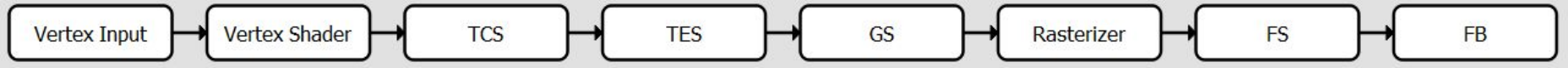

```
Program& programTexturedGeometry = app->programs[app->texturedGeometryProgramIdx];
glUseProgram(programTexturedGeometry.handle);
```

TODO: Let’s paint a textured quad



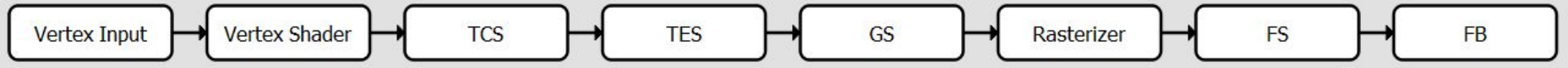

```
glBindVertexArray(app->vao);
```

TODO: Let’s paint a textured quad



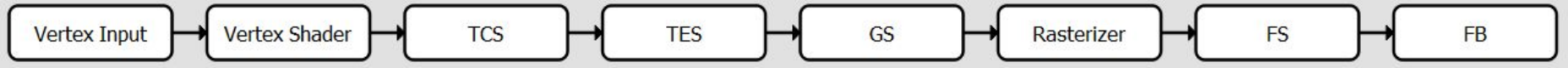

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

TODO: Let’s paint a textured quad



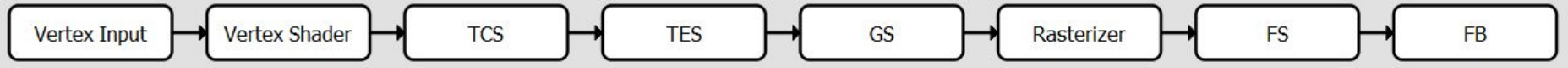

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

TODO: Let’s paint a textured quad



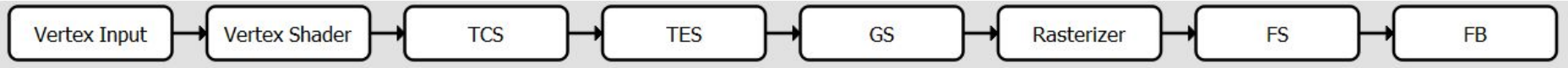

```
glUniform1i(app->programUniformTexture, 0);

glActiveTexture(GL_TEXTURE0);
GLuint textureHandle = app->textures[app->diceTexIdx].handle;
glBindTexture(GL_TEXTURE_2D, textureHandle);
```

TODO: Let’s paint a textured quad




```
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0);
```

TODO: Let’s paint a textured quad




```
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glViewport(0, 0, app->displaySize.x, app->displaySize.y);

Program& programTexturedGeometry = app->programs[app->texturedGeometryProgramIdx];
glUseProgram(programTexturedGeometry.handle);
glBindVertexArray(app->vao);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glUniform1i(app->programUniformTexture, 0);
glActiveTexture(GL_TEXTURE0);
GLuint textureHandle = app->textures[app->diceTexIdx].handle;
glBindTexture(GL_TEXTURE_2D, textureHandle);

glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0);

glBindVertexArray(0);
glUseProgram(0);
```


THAT'S COOL
YOU KIDS HAVE FUN