Shaders

GLSL openGL Shader Language

# version header e.g. 330 => openGL 3.30

input variables e.g. vec3

output variables (global) e.g. vec2

main function():

uv = vec2(1,1)

Vertex shader => have to specified the position : gl_Position()

Fragment shader => gl_FragColor(). have to specified the color generated

from resterization : out vec4 color ; (r,g,b,a)

explicit defined. e.g. color-out = openGL > 4.2

gl-FragColor = ...... let the program to assign.

interfacematching: variable name, data type, to match between the

output of previous shader and input of the next

openGLObject : using shader object.

1) create a shader :

GLuint vertID = glCreateShader(GL.VERTEX)

create a shader at <u>serverside</u>

=> on costomer side, it is usually an id

returned, an unsigned int

2) source code length of src, in byte

glShaderSource(I[D], char* src, int length) giving the id of the shader object created. => list of id / pointer

For each shader, we have to repeat these steps.

Program : render in pipeline

1.) Create prog -> GLuint progID = glCreateProgram().

2) Attach shader -> glAttachShader(progID, shaderID)

3) Link the shaders -> glLinkProgram(progID)

glGetShader(shaderID, GL.INFOLOG.LENGTH)

4.) Detach and delete shader -> glDelete(ShaderID)

then, get data into the shader

glEnableVertexAttributeArray(index)

glVertexAttributePointer(attributeIndex, => used pointer attribute index

num => number of data per vertex

type => int / float / etc...

GL.FALSE => normalization, always false,

0

pointer ) => to the data

glBegin()

↕ drawcall

glEnd()

glPosition = ModelViewProjection * vertexPosition

ID = glGetUniformLocation(progID,

glUniform(ID, data)

glUniformedMatrix(ID, 1, False, MVP)

| | |
|---|---|
| glDrawArrays | glDrawArrays (GL.TRIANGLE, startIndex, numVertices) |
| glDrawElements | glDrawElements(prim, startID, num, indices) |

$\uparrow$ a row/array of vertices.