Computer Science Department

Sam Houston State University

COSC 4332 Computer Graphics

Modern OpenGL
3D Introduction & Models Loading

Dr. Khaled Rabieh

Outline

1. **Introduction to 3D in Modern OpenGL**
2. Passing values from vertex shader to fragment shader
3. Models Loading

Changes to Vertex Shader to enable 3D

The vertex shader Modifications

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec3 vertexColor;

// Output data ; will be interpolated for each fragment.
out vec3 fragmentColor;
// Values that stay constant for the whole mesh.
uniform mat4 MVP;

void main(){

    // Output position of the vertex, in clip space : MVP * position
    gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

    // The color of each vertex will be interpolated
    // to produce the color of each fragment
    fragmentColor = vertexColor;
}
```

MVP are matrices That are set from inside the program

3D View

- Multiply the projection* View* Model matrices
- The result is MVP

```
// Create and compile our GLSL program from the shaders
GLuint programID = LoadShaders( "TransformVertexShader.vertexshader", "ColorFragmentShader.fragmentshader" );

// Get a handle for our "MVP" uniform
GLuint MatrixID = glGetUniformLocation(programID, "MVP");

// Projection matrix : 45° Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 3.0f, 0.1f, 100.0f);
// Camera matrix
glm::mat4 View       = glm::lookAt(
                            glm::vec3(4,3,-3), // Camera is at (4,3,-3), in World Space
                            glm::vec3(0,0,0), // and looks at the origin
                            glm::vec3(0,1,0)  // Head is up (set to 0,-1,0 to look upside-down)
                       );
// Model matrix : an identity matrix (model will be at the origin)
glm::mat4 Model      = glm::mat4(1.0f);
// Our ModelViewProjection : multiplication of our 3 matrices
glm::mat4 MVP        = Projection * View * Model; // Remember, matrix multiplication is the other way around
```

Vertex Attribute Array

```
static const GLfloat g_vertex_buffer_data[] = {
    -1.0f,-1.0f,-1.0f,
    -1.0f,-1.0f, 1.0f,
    -1.0f, 1.0f, 1.0f,
     1.0f, 1.0f,-1.0f,
    -1.0f,-1.0f,-1.0f,
    -1.0f, 1.0f,-1.0f,
     1.0f,-1.0f, 1.0f,
    -1.0f,-1.0f,-1.0f,
     1.0f,-1.0f,-1.0f,
     1.0f, 1.0f,-1.0f,
     1.0f,-1.0f,-1.0f,
    -1.0f,-1.0f,-1.0f,
    -1.0f,-1.0f,-1.0f,
    -1.0f, 1.0f, 1.0f,
    -1.0f, 1.0f,-1.0f,
     1.0f,-1.0f, 1.0f,
    -1.0f,-1.0f, 1.0f,
    -1.0f,-1.0f,-1.0f,
    -1.0f, 1.0f, 1.0f,
    -1.0f,-1.0f, 1.0f,
     1.0f,-1.0f, 1.0f,
     1.0f, 1.0f, 1.0f,
     1.0f,-1.0f,-1.0f,
     1.0f, 1.0f,-1.0f,
     1.0f,-1.0f,-1.0f,
     1.0f, 1.0f, 1.0f,
     1.0f,-1.0f, 1.0f,
     1.0f, 1.0f, 1.0f,
     1.0f, 1.0f,-1.0f,
    -1.0f, 1.0f,-1.0f,
     1.0f, 1.0f, 1.0f,
    -1.0f, 1.0f,-1.0f,
    -1.0f, 1.0f, 1.0f,
     1.0f, 1.0f, 1.0f,
```

```
GLuint vertexbuffer;
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
```

Color Attribute Array

```
static const GLfloat g_color_buffer_data[] = {
    0.583f,  0.771f,  0.014f,
    0.609f,  0.115f,  0.436f,
    0.327f,  0.483f,  0.844f,
    0.822f,  0.569f,  0.201f,
    0.435f,  0.602f,  0.223f,
    0.310f,  0.747f,  0.185f,
    0.597f,  0.770f,  0.761f,
    0.559f,  0.436f,  0.730f,
    0.359f,  0.583f,  0.152f,
    0.483f,  0.596f,  0.789f,
    0.559f,  0.861f,  0.639f,
    0.195f,  0.548f,  0.859f,
    0.014f,  0.184f,  0.576f,
    0.771f,  0.328f,  0.970f,
    0.406f,  0.615f,  0.116f,
    0.676f,  0.977f,  0.133f,
    0.971f,  0.572f,  0.833f, GLuint colorbuffer;
    0.140f,  0.616f,  0.489f, glGenBuffers(1, &colorbuffer);
    0.997f,  0.513f,  0.064f, glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
    0.945f,  0.719f,  0.592f, glBufferData(GL_ARRAY_BUFFER, sizeof(g_color_buffer_data), g_color_buffer_data, GL_STATIC_DRAW);
    0.543f,  0.021f,  0.978f,
    0.279f,  0.317f,  0.505f,
    0.167f,  0.620f,  0.077f,
    0.347f,  0.857f,  0.137f,
    0.055f,  0.953f,  0.042f,
    0.714f,  0.505f,  0.345f,
    0.783f,  0.290f,  0.734f,
    0.722f,  0.645f,  0.174f,
    0.302f,  0.455f,  0.848f,
    0.225f,  0.587f,  0.040f,
    0.517f,  0.713f,  0.338f,
    0.053f,  0.959f,  0.120f,
    0.393f,  0.621f,  0.362f,
    0.673f,  0.211f,  0.457f,
```

The program Loop

```
// 1rst attribute buffer : vertices
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glVertexAttribPointer(
    0,                  // attribute. No particular reason for 0, but must match the layout in the shader.
    3,                  // size
    GL_FLOAT,           // type
    GL_FALSE,           // normalized?
    0,                  // stride
    (void*)0            // array buffer offset
);
```

```
// 2nd attribute buffer : colors
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glVertexAttribPointer(
    1,                                // attribute. No particular reason for 1, but must match the layout in the shader.
    3,                                // size
    GL_FLOAT,                         // type
    GL_FALSE,                         // normalized?
    0,                                // stride
    (void*)0                          // array buffer offset
);
```

Vertex Shader

The vertex shader Modifications

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec3 vertexColor;

// Output data ; will be interpolated for each fragment.
out vec3 fragmentColor;
// Values that stay constant for the whole mesh.
uniform mat4 MVP;

void main(){

    // Output position of the vertex, in clip space : MVP * position
    gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

    // The color of each vertex will be interpolated
    // to produce the color of each fragment
    fragmentColor = vertexColor;
}
```

Will be forwarded to the fragment shader

The Fragment Shader

```
#version 330 core

// Interpolated values from the vertex shaders
in vec3 fragmentColor;

// Ouput data
out vec3 color;

void main(){

    // Output color = color specified in the vertex shader,
    // interpolated between all 3 surrounding vertices
    color = fragmentColor;

}
```


Fragment color is received here from the vertex shader

Life Cycle

How to get attributes array from the program passing through vertex shader to the fragment shader ?

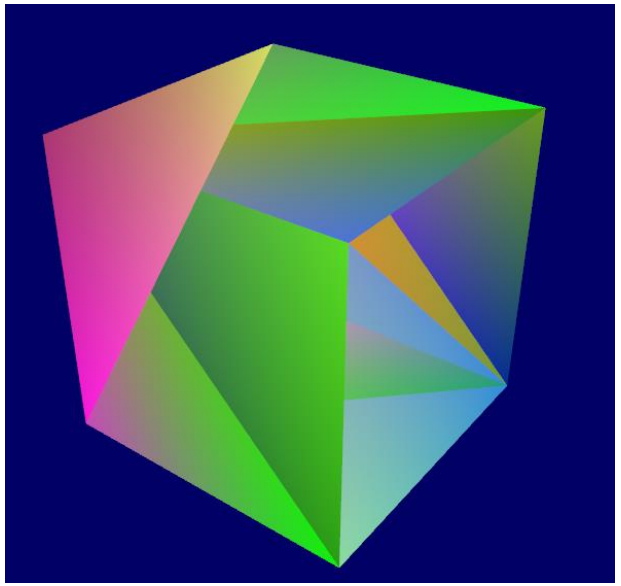
The program → The vertex shader → The fragement shader

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec3 vertexColor;

// Output data ; will be interpolated for each fragment.
out vec3 fragmentColor;
// Values that stay constant for the whole mesh.
uniform mat4 MVP;

void main(){

    // Output position of the vertex, in clip space : MVP * pos
    gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

    // The color of each vertex will be interp
    // to produce the color of each fragm
    fragmentColor = vertexColor;
}
```

```
#version 330 core

// Interpolated values from the vertex shaders
in vec3 fragmentColor;

// Ouput data
out vec3 color;

void main(){

    // Output color = color specified in the vertex shader,
    // interpolated between all 3 surrounding vertices
    color = fragmentColor;

}
```

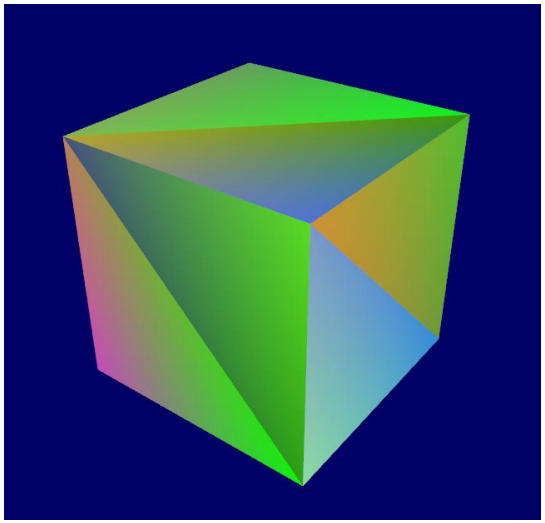
```
// 2nd attribute buffer : colors
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glVertexAttribPointer(
    1,                                // at
    3,                                // si
    GL_FLOAT,                         // ty
    GL_FALSE,                         // no
    0,                                // st
    (void*)0                          // ar
);
```

Enable GL_DEPTH_TEST



Every time you want to write a fragment, you first check if you should (i.e the new fragment is closer than the previous one).

```
// Enable depth test
glEnable(GL_DEPTH_TEST);
// Accept fragment if it closer to the camera than the former one
glDepthFunc(GL_LESS);
```



Textures

Using SOIL, Create a texture, bind it, fill it, and configure it.

```
GLuint loadtextures(char* filename)
{
    image = SOIL_load_image(filename, &width, &height, 0, SOIL_LOAD_RGB);
    GLuint textureId;
    glGenTextures(1, &textureId); //Make room for our texture
    glBindTexture(GL_TEXTURE_2D, textureId);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
    SOIL_free_image_data(image);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glGenerateMipmap(GL_TEXTURE_2D);
    return textureId;
}
```

The Vertex Shader

To be filled from the program uv attributes array

Out to the fragment shader

Assign the value of vertexUV to UV

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec2 vertexUV;

// Output data ; will be interpolated for each fragment.
out vec2 UV;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;

void main(){

    // Output position of the vertex, in clip space : MVP * position
    gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

    // UV of the vertex. No special space for this one.
    UV = vertexUV;
}
```

The Fragment Shader

The fragment shader needs

- UV coordinates
- A “sampler2D” in order to know which texture to access (you can access several texture in the same shader)
- texture(), which gives back a (R,G,B,A) vec4.

```
#version 330 core

// Interpolated values from the vertex shaders
in vec2 UV;

// Ouput data
out vec3 color;

// Values that stay constant for the whole mesh.
uniform sampler2D myTextureSampler;

void main(){

    // Output color = color of the texture at the specified UV
    color = texture( myTextureSampler, UV ).rgb;
}
```

UV Attributes Array

```
static const GLfloat g_uv_buffer_data[] = {
    0.000059f, 1.0f-0.000004f,
    0.000103f, 1.0f-0.336048f,
    0.335973f, 1.0f-0.335903f,
    1.000023f, 1.0f-0.000013f,
    0.667979f, 1.0f-0.335851f,
    0.999958f, 1.0f-0.336064f,
    0.667979f, 1.0f-0.335851f,
    0.336024f, 1.0f-0.671877f,
    0.667969f, 1.0f-0.671889f,
    1.000023f, 1.0f-0.000013f,
    0.668104f, 1.0f-0.000013f,
    0.667979f, 1.0f-0.335851f,
    0.000059f, 1.0f-0.000004f,
    0.335973f, 1.0f-0.335903f,
    0.336098f, 1.0f-0.000071f,
    0.667979f, 1.0f-0.335851f,
    0.335973f, 1.0f-0.335903f,
    0.336024f, 1.0f-0.671877f,
    1.000004f, 1.0f-0.671847f,
    0.999958f, 1.0f-0.336064f,
    0.667979f, 1.0f-0.335851f,
    0.668104f, 1.0f-0.000013f,
    0.335973f, 1.0f-0.335903f,
    0.667979f, 1.0f-0.335851f,
    0.335973f, 1.0f-0.335903f,
    0.668104f, 1.0f-0.000013f,
    0.336098f, 1.0f-0.000071f,
    0.000103f, 1.0f-0.336048f,
    0.000004f, 1.0f-0.671870f,
    0.336024f, 1.0f-0.671877f,
    0.000103f, 1.0f-0.336048f,
    0.336024f, 1.0f-0.671877f,
    0.335973f, 1.0f-0.335903f,
    0.667969f, 1.0f-0.671889f,
    1.000004f, 1.0f-0.671847f,
    0.667979f, 1.0f-0.335851f
};
```

TWO Buffer Arrays

```
GLuint vertexbuffer;
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);

GLuint uvbuffer;
glGenBuffers(1, &uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_uv_buffer_data), g_uv_buffer_data, GL_STATIC_DRAW);

 // Load the texture using any two methods
 //GLuint Texture = loadBMP_custom("uvtemplate.bmp");
 GLuint Texture = loadtextures("wooden.jpg");

 // Get a handle for our "myTextureSampler" uniform
 GLuint TextureID  = glGetUniformLocation(programID, "myTextureSampler");
```

In the program Loop

```
 // Bind our texture in Texture Unit 0
 glActiveTexture(GL_TEXTURE0);
 glBindTexture(GL_TEXTURE_2D, Texture);
 // Set our "myTextureSampler" sampler to use Texture Unit 0
 glUniform1i(TextureID, 0);
```

Models Loading

- Hardcoding the vertices directly in the source code is not very handy.
- **OBJ file format**, which is both very simple and very common.

```
# Blender3D v249 OBJ File: untitled.blend
# www.blender3d.org
mtllib cube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 0.748573 0.750412
vt 0.749279 0.501284
vt 0.999110 0.501077
vt 0.999455 0.750380
vt 0.250471 0.500702
```

- # is a comment, just like // in C++
- usemtl and mtllib describe the look of the model. We won't use this in this tutorial.
- v is a vertex
- vt is the texture coordinate of one vertex

Model Loading

```
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
usemtl Material_ray.png
s off
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
```

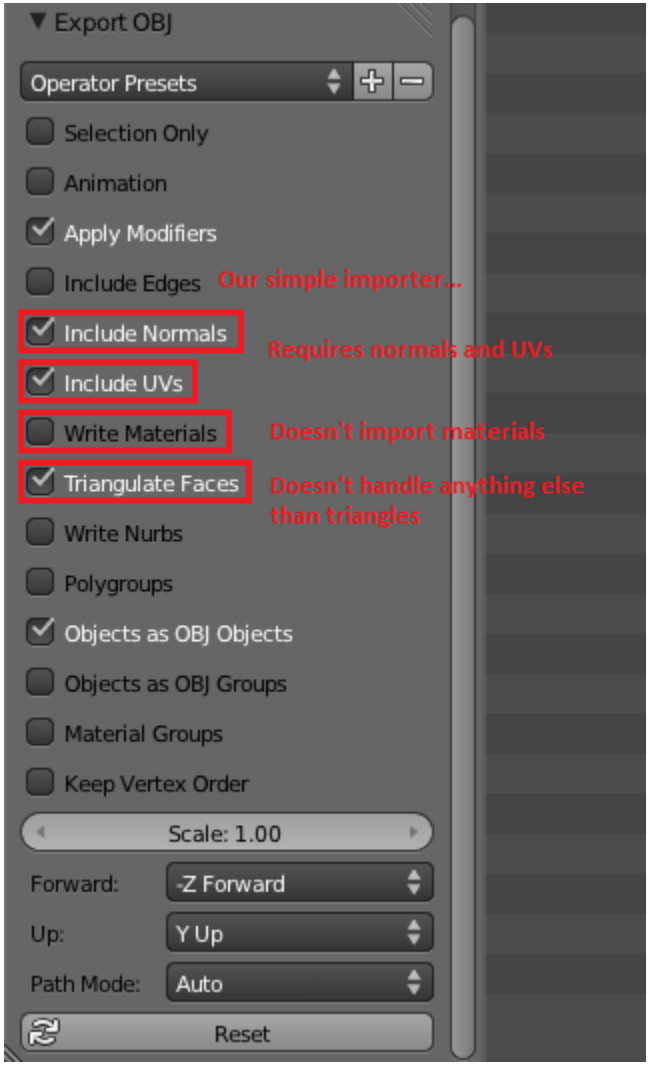
- vn is the normal of one vertex
- f is a face/primitive/triangle

- The first vertex of the triangle is 5/1/1
- Use the 5th vertex
- Use the 1st texture co-ordinate
- Use the 1st normal
- These arrays are 1 based and not zero based

Generating/Downloading Models

Blender or Maya software

https://www.raywenderlich.com/48293/how-to-export-blender-models-to-opengl-es-part-1

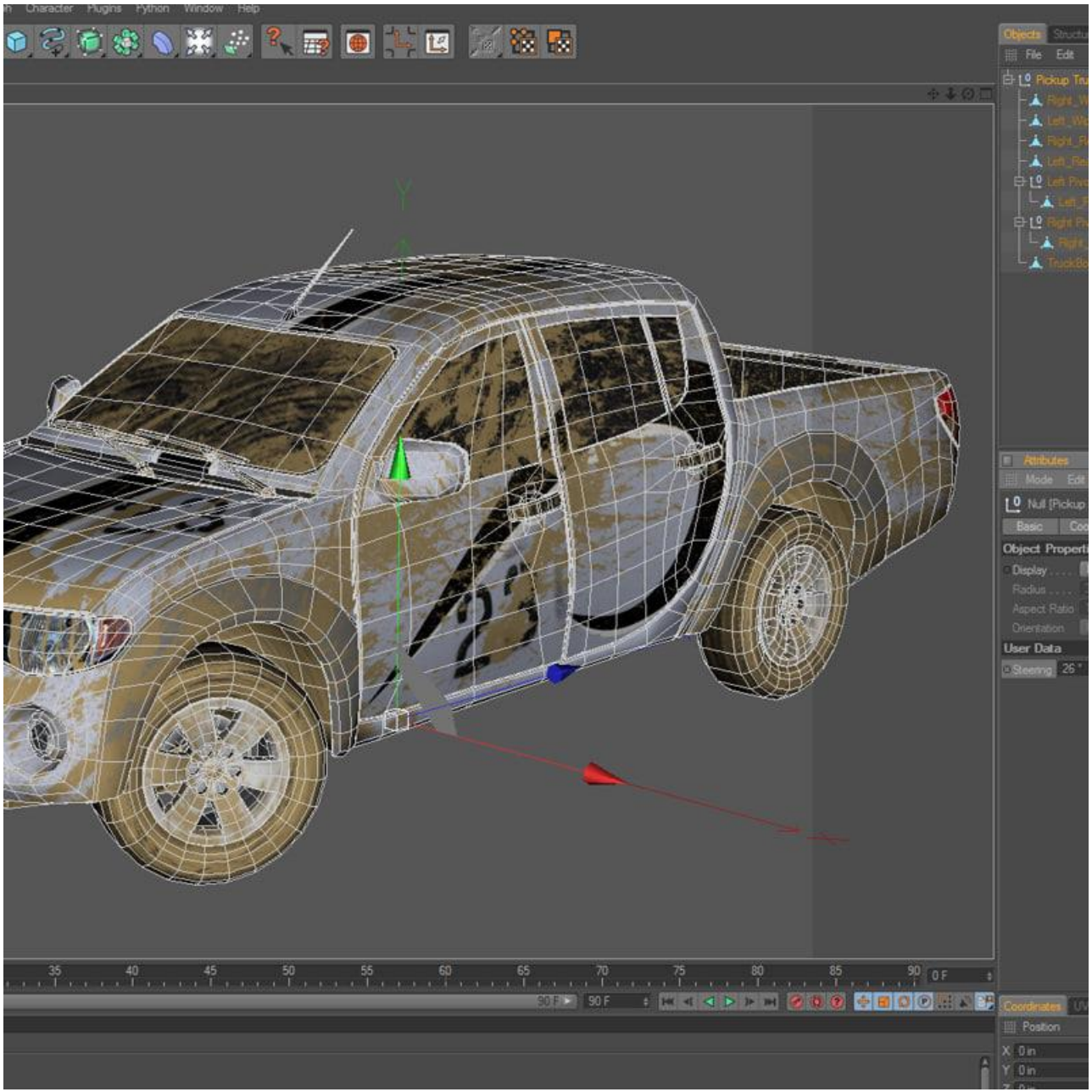


Maya can be downloaded from :
ttps://www.autodesk.com/education/free-software/maya

OR just download an obj model from the internet

Free Models can be found in https://www.turbosquid.com/

Modeling In Blender Example

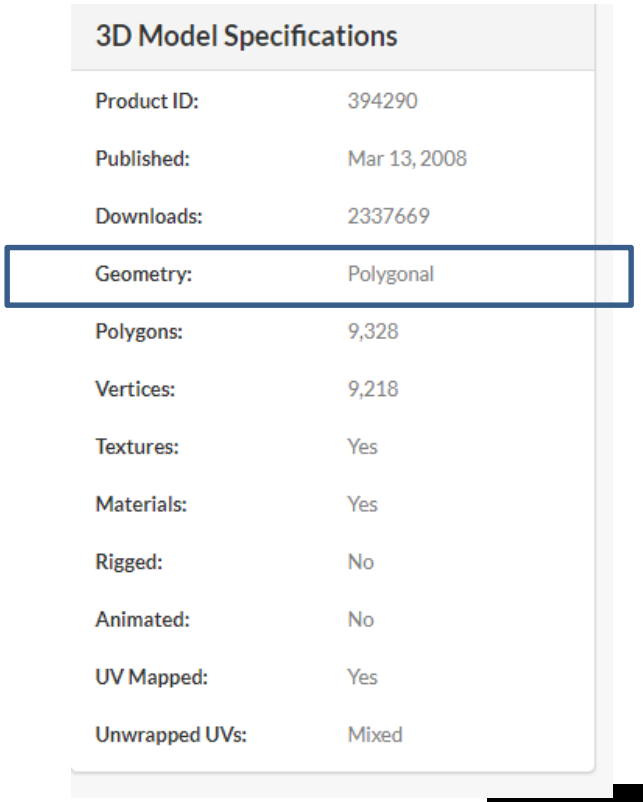


Warning !

- Any Obj model who uses a quad as a primitive will not work with Modern openGL
- GL_QUADS was deprecated in version 3.0 and removed in version 3.1.
- Obviously this is all irrelevant if you create a compatibility context.

```
f 9200/11517/15498 9197/11515/15493 9212/11523/15509
9213/11525/15514
s 6
f 9199/11516/15512 9201/11536/15515 9215/11526/15516
9216/11524/15513
```

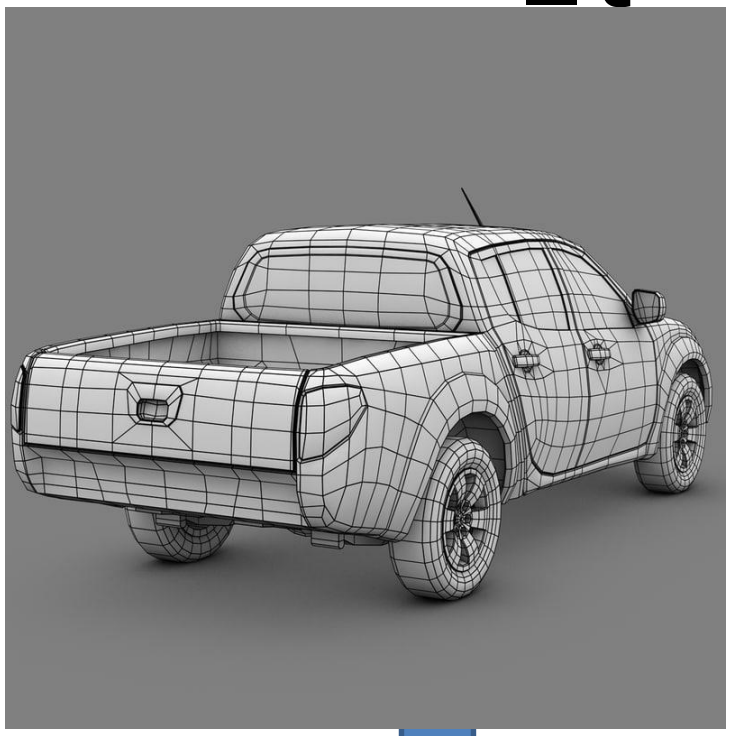
```
// Draw the triangle !
glDrawArrays(GL_TRIANGLE_STRIP, 0, vertices.size() );
```



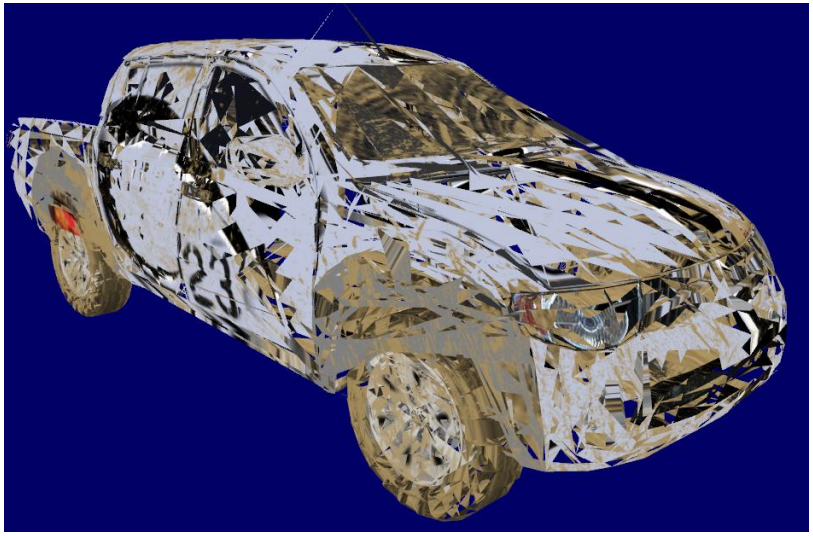
3D Model Specifications

Product ID:	394290
Published:	Mar 13, 2008
Downloads:	2337669
Geometry:	Polygonal
Polygons:	9,328
Vertices:	9,218
Textures:	Yes
Materials:	Yes
Rigged:	No
Animated:	No
UV Mapped:	Yes
Unwrapped UVs:	Mixed

GL_Quads is not Supported



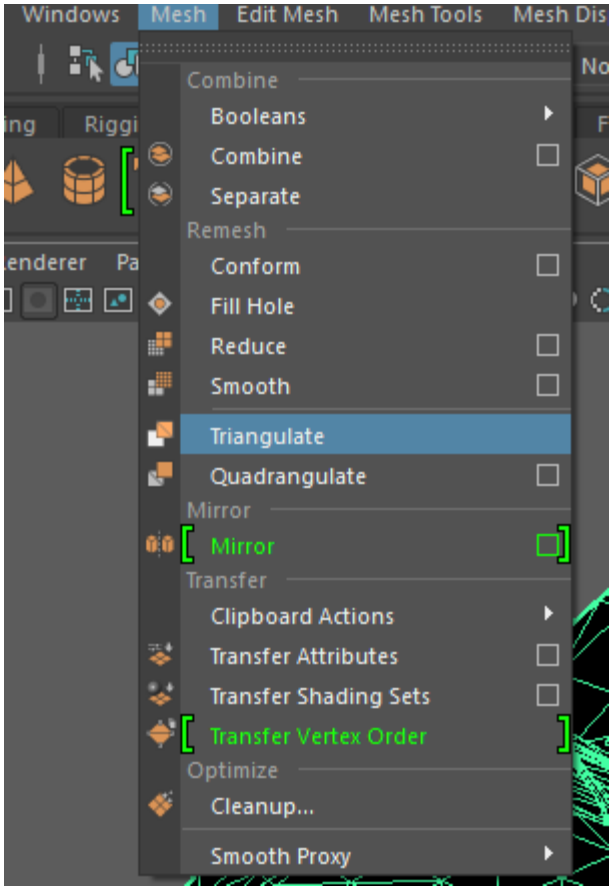
Should look like this

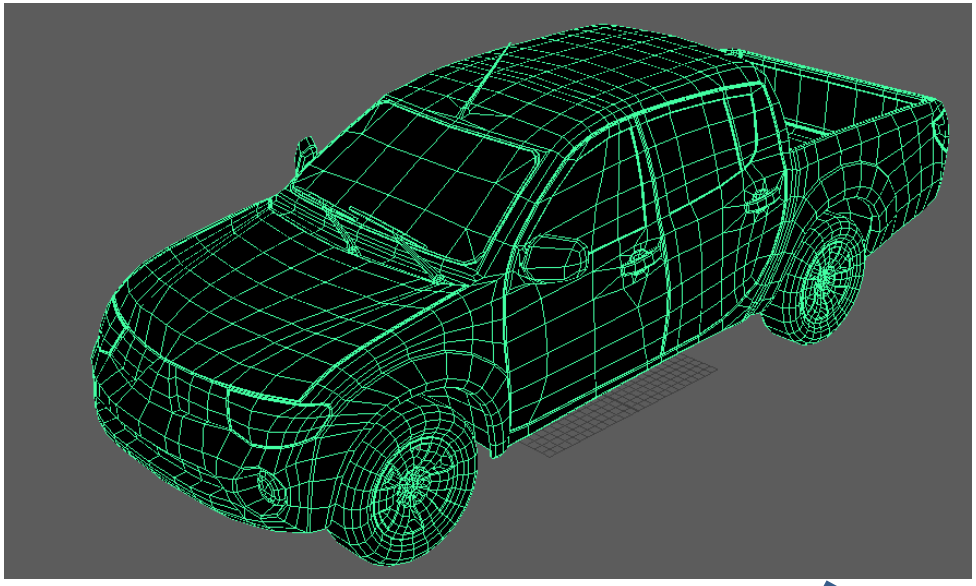


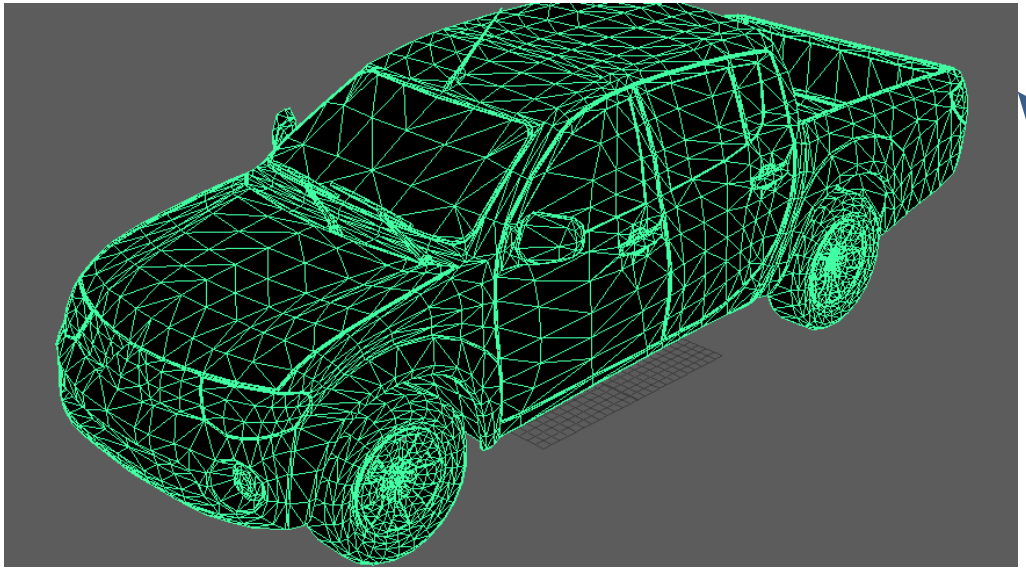
https://www.turbosquid.com/FullPreview/Index.cfm/ID/394290

The Fix

- AutoDesk Maya converts an obj with quads facets to triangulates facets







Using objloader Class

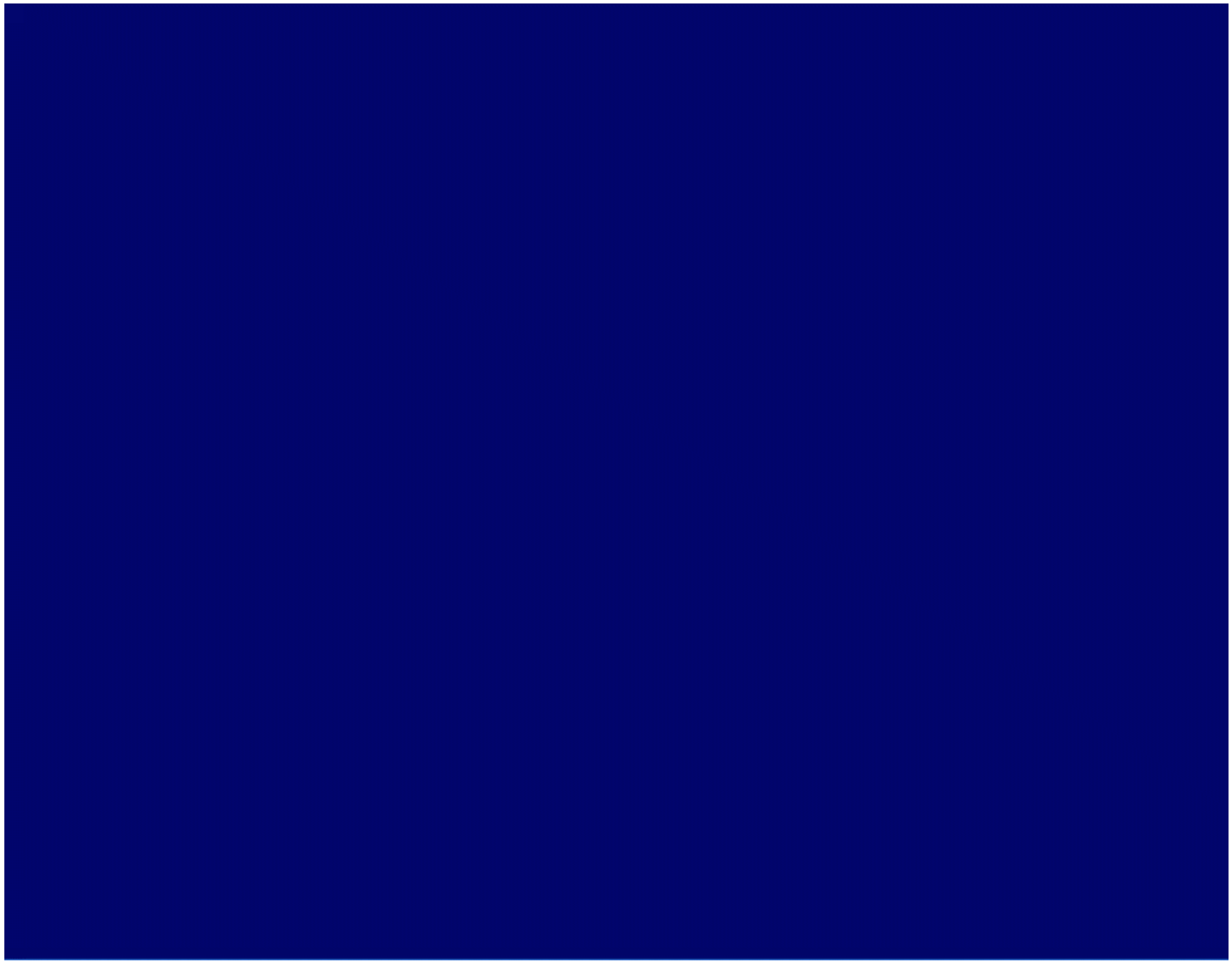
```
// Read our .obj file
std::vector< glm::vec3 > vertices;
std::vector< glm::vec2 > uvs;
std::vector< glm::vec3 > normals; // Won't be used at the moment.
bool res = loadOBJ("cube.obj", vertices, uvs, normals);
```

and give your vectors to OpenGL instead of your arrays :

```
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3), &vertices[0], GL_STATIC_DRAW);
```

Your turn

1- Load The Model named “FREOBJ.obj” to be beside the truck.



Thank You

Questions

Khaled Rabieh