ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

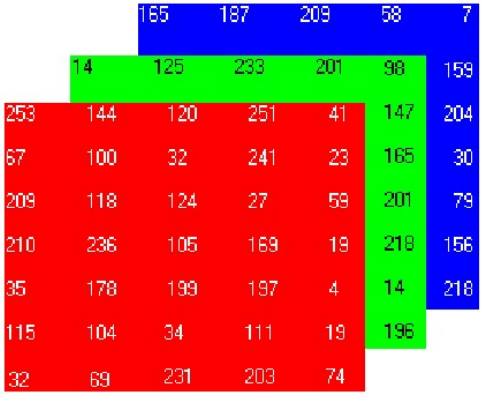
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εισαγωγή στα Γραφικά Υπολογιστών

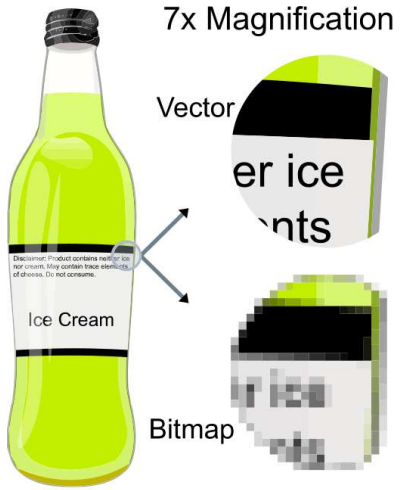
Εργαστήριο 1

Γραφικά

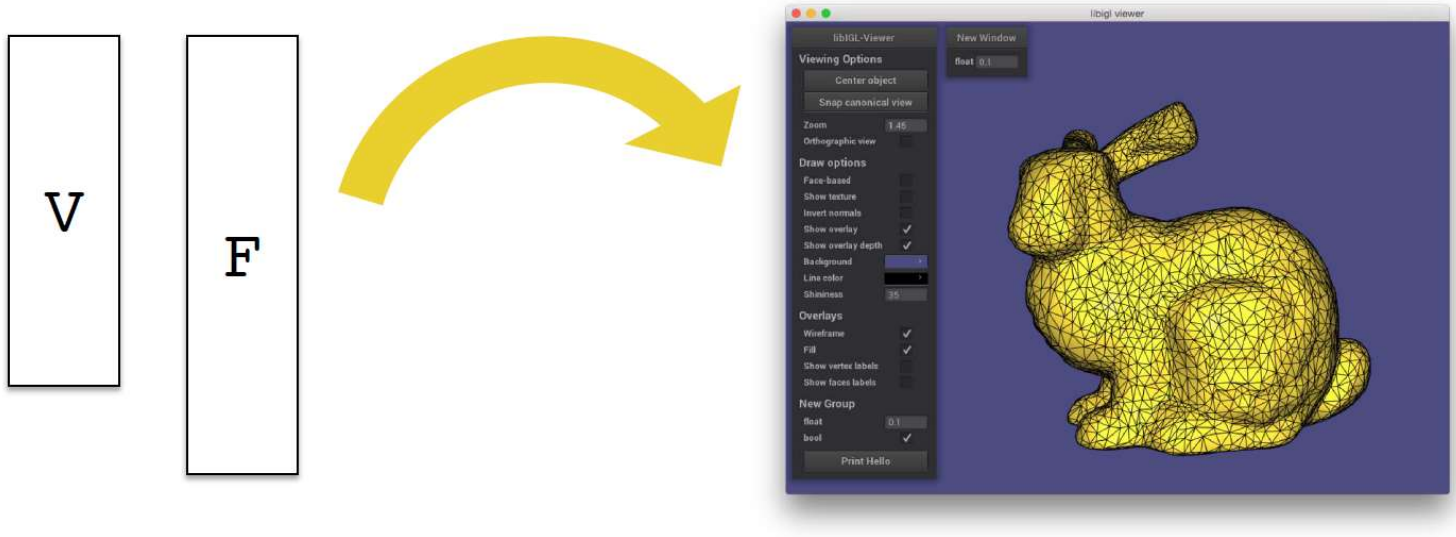
- **Raster Graphics**
  - Πίνακας με μια θέση για κάθε pixel της εικόνας
  - Κανάλια Red, Green, Blue με τιμές στο [0, 255]



- **Vector Graphics**
  - Αποθηκεύονται σημεία στον χώρο και οι γεωμετρικές σχέσεις μεταξύ τους
  - Η εικόνα μπορεί να αναπαραχθεί για οποιαδήποτε ανάλυση
  - Αρχεία PDF

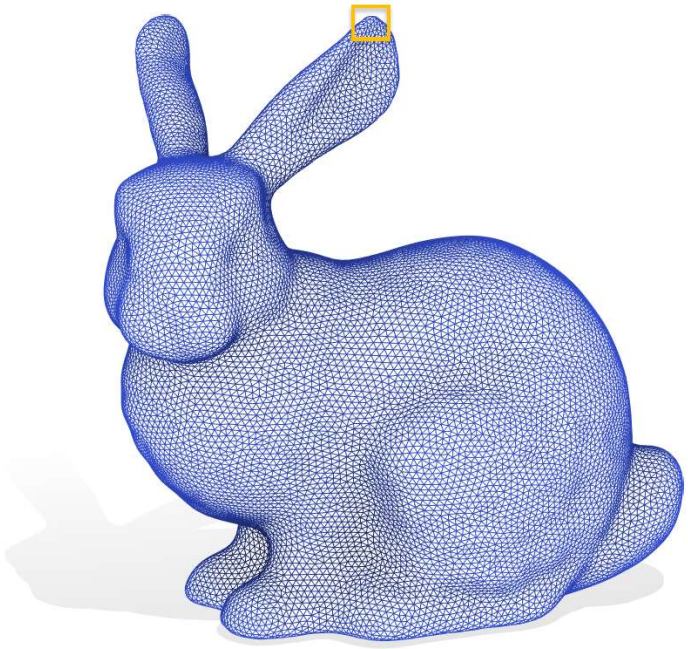


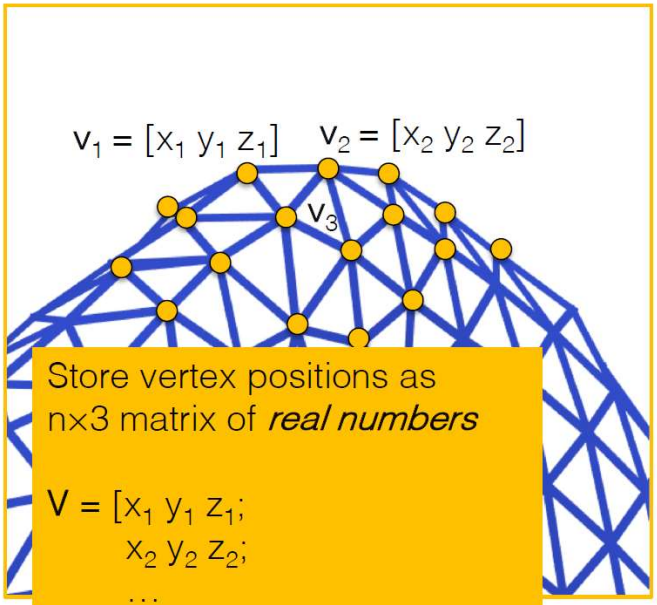
Meshes


V
F
libigl viewer
libIGL-Viewer
Viewing Options
Center object
Snap canonical view
Zoom
Orthographic view
Draw options
Face-based
Show texture
Invert normals
Show overlay
Show overlay depth
Background
Line color
Shininess
Overlays
Wireframe
Fill
Show vertex labels
Show faces labels
New Group
float
bool
Print Hello
New Window
float

Meshes

Triangle meshes discretize surfaces…





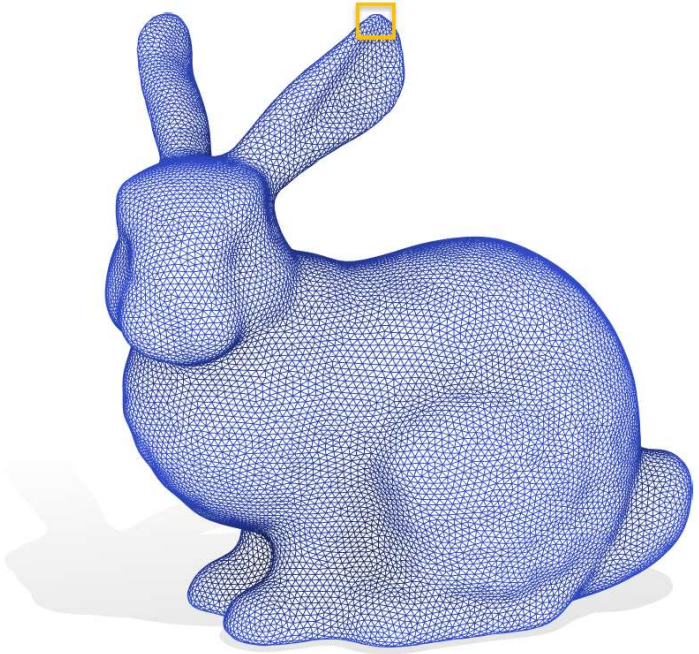
Store vertex positions as
n×3 matrix of *real numbers*

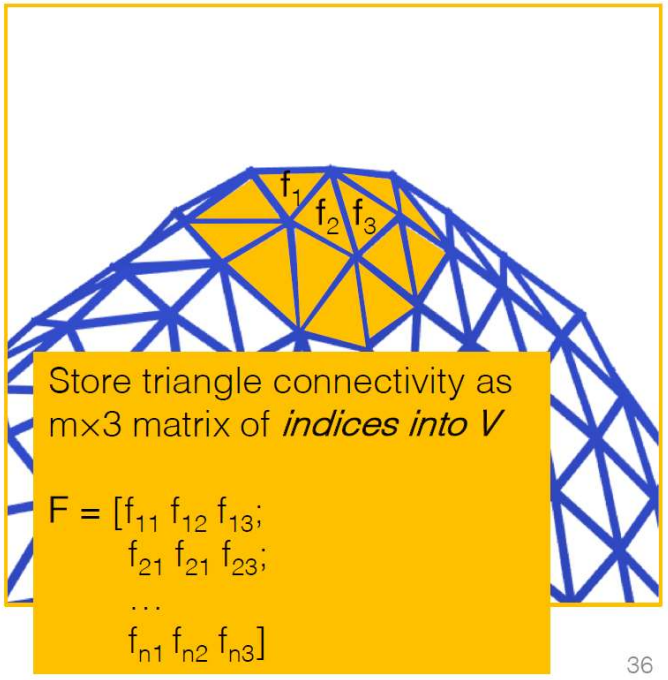
$$V = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \dots & & \\ x_n & y_n & z_n \end{bmatrix}$$

28

Meshes

Triangle meshes discretize surfaces…





36

OpenGL

- **API**: Application Programming Interface
- Αναπτύσσεται και συντηρείται από το Khronos Group

- Για τους κατασκευαστές των καρτών γραφικών: Μια σειρά από specifications που καλούνται να υλοποιήσουν

- Για εμάς: Μια σειρά από συναρτήσεις, built-in μεταβλητές και τύπους δεδομένων

OpenGL

- Εξελίσσεται παράλληλα με τις δυνατότητες του hardware

- Επιπλέον δυνατότητες που υπάρχουν σε καινούριες κάρτες δε χρειάζεται να ‘περιμένουν’ την επόμενη έκδοση OpenGL

- Extensions:
  - NV (Nvidia)
  - EXT (υλοποιείται από πολλούς κατασκευαστές)
  - ARB (Architecture Review Board approved)

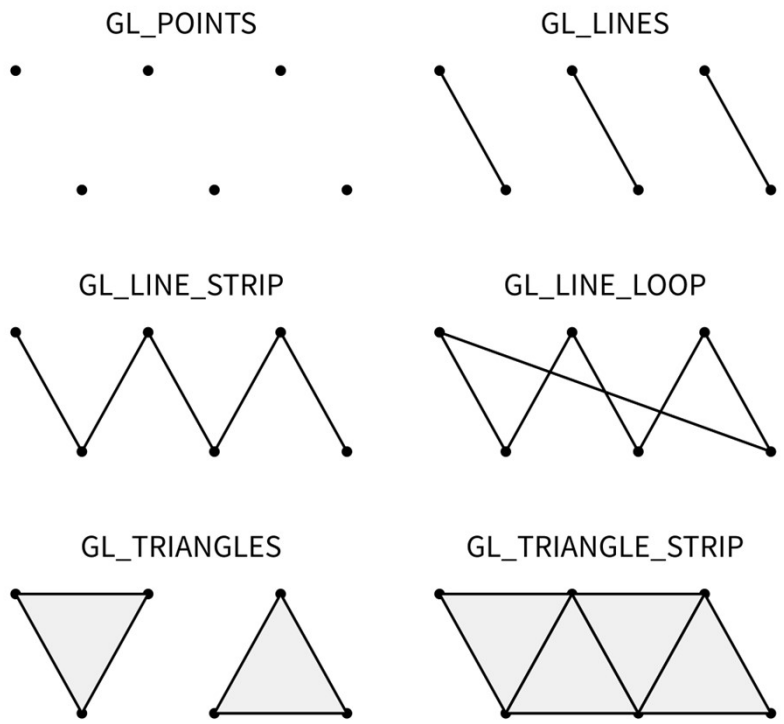
OpenGL

- Πρώτη έκδοση 1.0: Fixed Pipeline
  - Προκαθορισμένος τρόπος απόδοσης (rendering) των γραφικών
- Έκδοση 2.0: Ενσωμάτωση της γλώσσας GLSL (OpenGL Shading Language)
  - Δυνατότητες προγραμματισμού των διαδικασιών που γίνονται στην κάρτα γραφικών
- Έκδοση 3.2: Διαχωρισμός των Core και Compatibility Profiles
- Τελευταία έκδοση μέχρι σήμερα: 4.6

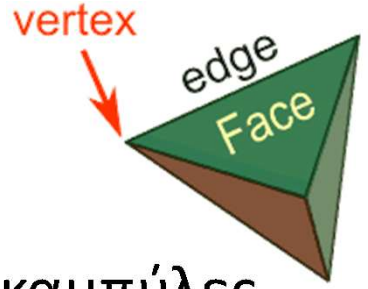
OpenGL

- OpenGL State (και OpenGL **Context**): Μια σειρά από μεταβλητές που δίνουν οδηγίες στην OpenGL για το πώς θα λειτουργήσει

- Αλλάζουμε το state μέσα από ειδικές συναρτήσεις, βάζουμε δεδομένα σε ειδικούς buffers και κάνουμε render με το παρόν context



Βασικοί Ορισμοί



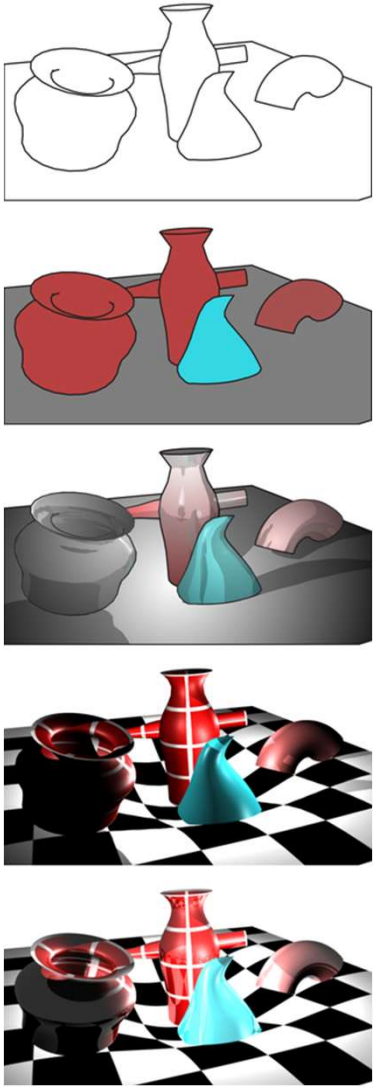
- **Vertices** (κορυφές): σημεία στα οποία συναντιούνται δύο καμπύλες, γραμμές ή έδρες
- **Primitive**: το πιο απλό γεωμετρικό αντικείμενο που σχεδιάζεται από την OpenGL
- **Pixel**: το μικρότερο στοιχείο μιας εικόνας ή μιας οθόνης
- **Fragment**: όλη η πληροφορία που είναι απαραίτητη ώστε να γίνει render ένα pixel από την OpenGL

Βασικοί Ορισμοί

- **Shaders**: προγράμματα που τρέχουν στην κάρτα γραφικών, έχουν διάφορες χρήσεις με συνηθέστερη τον φωτισμό (shading)

Βασικοί Ορισμοί

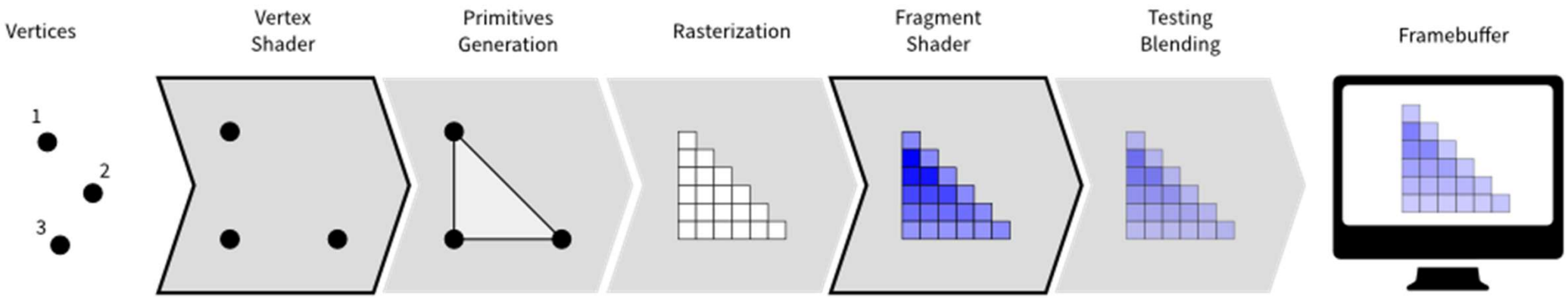
- **Rendering**: η διαδικασία αυτόματης παραγωγής μιας εικόνας (φωτορεαλιστικής ή όχι) από μια 2Δ ή 3Δ σκηνή (μια ομάδα από 2Δ ή 3Δ αντικείμενα).
- **Framebuffer**: ο buffer ο οποίος περιέχει το bitmap που προβάλλεται στην οθόνη, περιέχει τα δεδομένα ενός frame
- **Viewport**: το μέρος του παραθύρου στο οποίο εμφανίζονται τα γραφικά



Εικόνα του Maximilian Schönherr

OpenGL Pipeline

1) Μετατροπή 3Δ συντεταγμένων σε 2Δ συντεταγμένες
2) Μετατροπή 2Δ συντεταγμένων σε χρωματισμένα pixel στην οθόνη



OpenGL

- Η OpenGL είναι ανεξάρτητη του λειτουργικού συστήματος
- Δεν παρέχει την δυνατότητα να δημιουργήσουμε παράθυρο, να ορίσουμε Context και να χειριστούμε την είσοδο του χρήστη γιατί εξαρτώνται από το λειτουργικό σύστημα
- Οι λειτουργίες αυτές θα γίνουν με τη βοήθεια βιβλιοθήκης GLFW

OpenGL

- Η OpenGL είναι απλά ένα specification, η υλοποίηση εξαρτάται από τον κάθε κατασκευαστή και βρίσκεται στον driver της κάρτας γραφικών

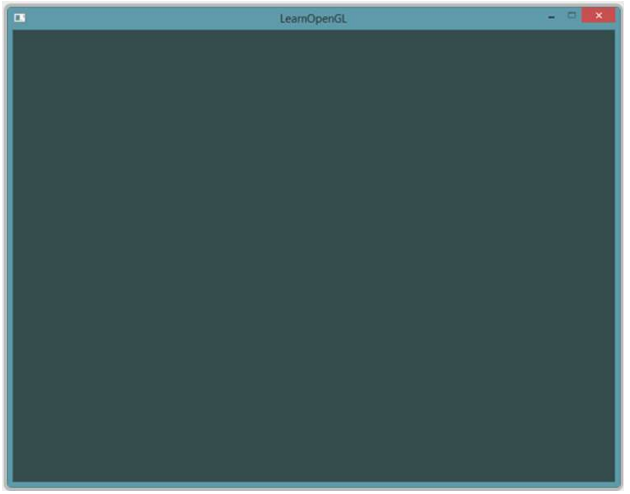
- Λόγω του πλήθους των drivers, η θέση κάθε συνάρτησης δεν είναι γνωστή κατά το compile-time και πρέπει να βρεθεί κατά το run-time

- Για να φορτώσουμε αυτόματα όλες τις διαθέσιμες συναρτήσεις (και extrensions) θα χρησιμοποιήσουμε τη βιβλιοθήκη GLEW

Hello Window

- Για να εμφανίσουμε ένα παράθυρο, πρέπει να:

1) Αρχικοποιήσουμε την glfw,
2) Δημιουργήσουμε ένα παράθυρο,
3) Δημιουργήσουμε ένα OpenGL context,
4) Αρχικοποιήσουμε την glew,
5) Δημιουργήσουμε ένα OpenGL viewport,
6) Δημιουργήσουμε ένα rendering loop.



Hello window

1) Αρχικοποίηση της GLFW: glfwInit()

```
if (!glfwInit())
{
	std::cout << "Failed to initialize GLFW" << std::endl;
	return -1;
}
```

Τερματισμός GLFW πριν την έξοδο: glfwTerminate()

Hello window

2) και 3) Δημιουργία παραθύρου και Context: glfwCreateWindow()

```
GLFWwindow* window = glfwCreateWindow(800, 600, "LearnOpenGL",
NULL, NULL);

if (window == NULL)
{
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
}
//make the context that was created the current one
glfwMakeContextCurrent(window);
```

Hello window

4) Αρχικοποίηση της GLEW: glewInit()

```
if (glewInit() != GLEW_OK)
{
	std::cout << "Failed to initialize GLEW" << std::endl;
	glfwTerminate();
	return -1;
}
```

Hello window

5) Δημιουργία ενός viewport: glViewport(…)

- Ξεχωριστή συνάρτηση:

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

- Στη συνάρτηση main:

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
```

Hello window

6) Για να μείνει το παρράθυρο ανοιχτό, πρέπει να προσθέσουμε ένα *rendering loop:*

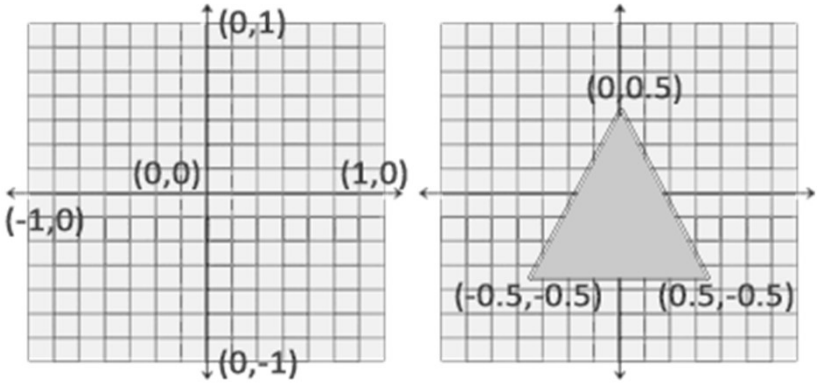
```
while(!glfwWindowShouldClose(window))
{
   //render scene
   glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
   glClear(GL_COLOR_BUFFER_BIT);

   //swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
   glfwSwapBuffers(window);
   glfwPollEvents();
}
```

Triangle Vertices

- **Normalized Device Coordinates(NDC)**: οι συντεταγμένες x, y και z με τιμές στο εύρος [-1.0, 1.0] θα εμφανιστούν στην οθόνη.

```
GLfloat vertices[] = {
    -0.5f, -0.5f, 0.0f,
     0.5f, -0.5f, 0.0f,
     0.0f,  0.5f, 0.0f
};
```



Vertex Buffer Object

```
//generate a unique buffer ID
GLuint triangle_vbo;
glGenBuffers(1, &triangle_vbo);

//bind the buffer to the context
glBindBuffer(GL_ARRAY_BUFFER, triangle_vbo);

//copy user-defined data into the currently bound buffer
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Vertex Shader

- Το πρόγραμμα του vertex shader είναι γραμμένο στη γλώσσα GLSL:

```
#version 330 core
layout (location = 0) in vec3 in_position;

void main()
{
    gl_Position = vec4(in_position.x, in_position.y, in_position.z, 1.0);
}
```

Vertex Shader

- Για να προσθέσουμε τον vertex shader στο πρόγραμμα, πρέπει να έχει τη μορφή ενός **const** string:

```
const char* vertexShaderSource =
"#version 330 \n"
" layout (location = 0) in vec3 in_position; \n"
"void main() \n"
"{ \n"
"    gl_Position = vec4(in_position.x, in_position.y, in_position.z, 1.0);\n"
"} \n";
```

Vertex Shader

```
GLuint vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);

//attach the shader source code to the shader object
glShaderSource(vertexShader, 1, & vertexShaderSource, NULL);

//compile the shader program
glCompileShader(vertexShader);
```

Fragment Shader

```
#version 330 core

void main()
{
    gl_FragColor = vec4(1.0, 0.5, 0.2, 1.0);
}
```

Fragment Shader

```
const char* fragmentShaderSource =
"#version 330 core \n"
"void main (void) \n"
"{ \n"
"     gl_FragColor = vec4(1.0, 0.5, 0.2, 1.0); \n"
"} \n";
```

Fragment Shader

```
GLuint fragmentShader;
fragmentShader  = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
```

Shader Program

- Για να χρησιμοποιήσουμε τους shaders που έχουμε, πρέπει να τους συνδέσουμε με ένα αντικείμενο shader program:

```
GLuint shaderProgram;
shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
```

Shader Program

- Αφού έχει γίνει το linking του προγράμματος, μπορούμε να το χρησιμοποιήσουμε όποτε θέλουμε να αποδώσουμε(render) ένα αντικείμενο:

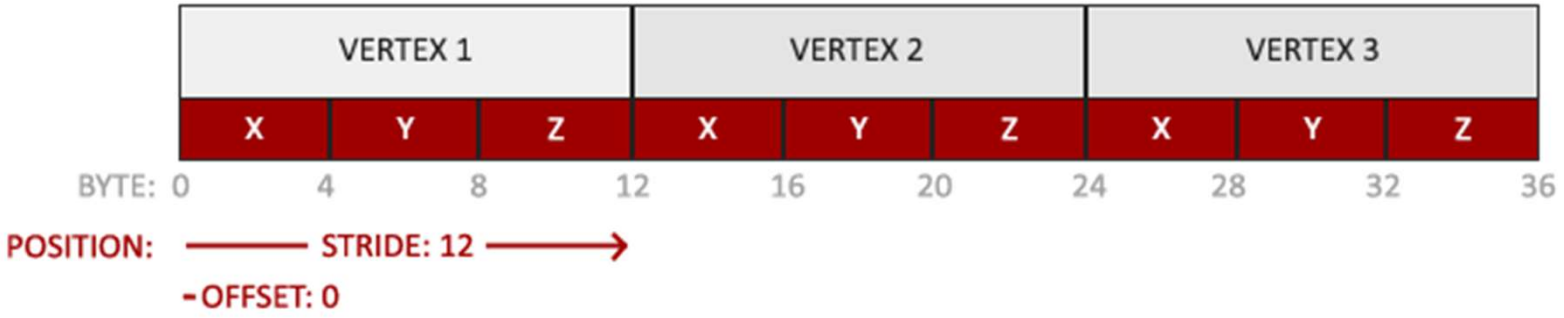
```
glUseProgram(shaderProgram);
```

- Μετά το linking, μπορούμε να διαγράψουμε τα αντικείμενα των shader:

```
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

Hello Triangle

- Τα δεδομένα μας βρίσκονται στο VBO, με αυτή τη μορφή:



Hello Triangle

```
//first bind the VBO…
glBindBuffer(GL_ARRAY_BUFFER, triangle_vbo);

//how to read the data in the  bound VBO
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GL_FLOAT), (void*)0);

//enable the attribute
glEnableVertexAttribArray(0);
```

Hello Triangle

- Rendering:

glDrawArrays(GL_TRIANGLES, 0, 3);

- 1ο: ο τύπος primitive (πχ GL_TRIANGLES, GL_POINTS, GL_LINE_STRIP)
- 2ο: ο πρώτος δείκτης του πίνακα με τις κορυφές
- 3ο: πόσες κορυφές θέλουμε να σχεδιάσουμε

Hello Triangle

- Άρα στο rendering loop θα γίνουν τα εξής:

```
glUseProgram(shaderProgram);

glBindBuffer(GL_ARRAY_BUFFER, triangle_vbo);

glDrawArrays(GL_TRIANGLES, 0, 3);
```

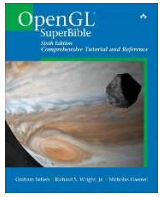
Στο σπίτι

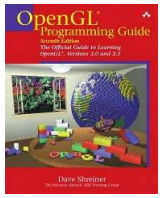
- Εγκαταστήστε το πρόγραμμα OpenGL Extensions Viewer: Δείτε ποιες εκδόσεις της OpenGL υποστηρίζονται από την κάρτα (ή κάρτες) γραφικών σας.

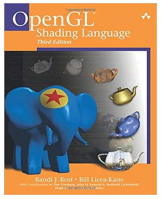
Κατεβάστε το από τη διεύθυνση:

http://realtech-vr.com/admin/glview

Books and Sites with OpenGL tutorials

- OpenGL Superbible 

- OpenGL Programming Guide 

- OpenGL Shading Language 

- Υλικό χρησιμοποιήθηκε από το site: learnopengl.com (έκδοση 3.3)