Podstawy Grafiki Komputerowej

Wykład 2: OpenGL

Andrzej Łukaszewski

Pracownia Grafiki Komputerowej
(LIGHT — Laboratory of Imaging and GrapHic Techniques)

Instytut Informatyki, Uniwersytet Wrocławski

14 października 2019

Wstęp: API: bitmap

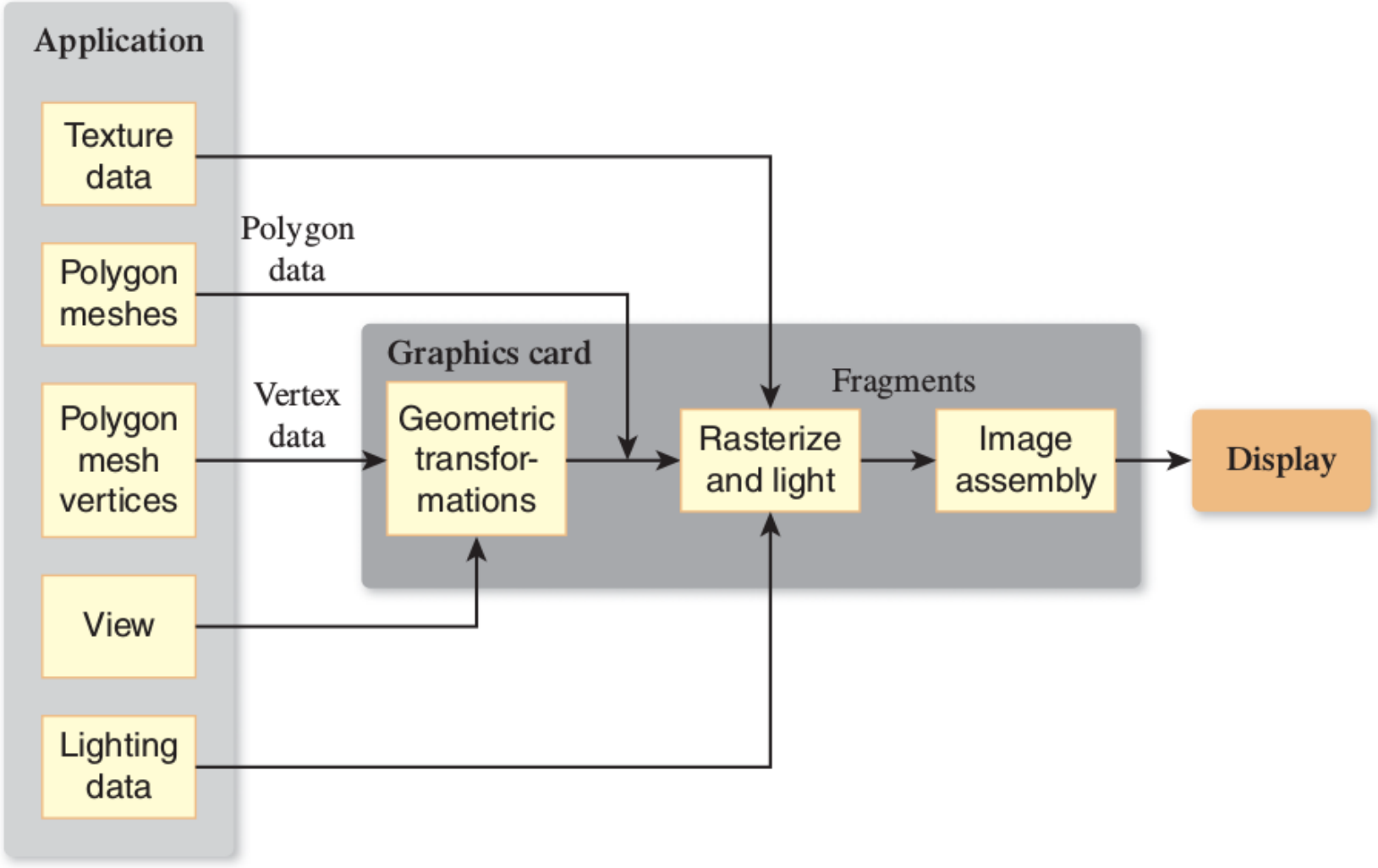
Grafika rastrowa: operacje na obszarze pamięci bufora ekranu

- w pamięci kolejne dane dla $X_{res} \cdot Y_{res}$ pikseli
- dla piksela: 1bit, 16bitow, 1,2,3,4,6,8,... bajtów (odcienie szarości/kolor RGB, liczby całkowite/zmiennoprzecinkowe)
- proste API: 4 funkcje: Open, Close, Put, Get:

```
1  int main() {
2     OpenGraphics(); // może podać w parametrach rozdzielczość?
3     color = GetPixel(0,0);
4     // rysujemy poziomy odcinek
5     for (int i=0; i<100; i++) PutPixel(i, 10, color);
6     // . . .  miejsce na interakcje z użytkownikiem
7     CloseGraphics();
8  }
```

Uwaga: padding

Wstęp: API: potok 3D



Potok graficzny w prostej wersji

OpenGL: Khronos konsorcjum

Konsorcjum promujące i rozwijające otwarte standardy wspierające API dla obsługi sprzętowej akceleracji grafiki i obliczeń (rok zał. 2000).
Wszystkie większe firmy: Intel, Nvidia, AMD, Apple, . . .

1. **1992** OpenGL — oryginalnie SGI
2. **2003** OpenGL ES 1.0(fixed)/2.0+(prog.)
3. **2008** OpenCL — obliczenia równoległe
4. **2011** WebGL — w przeglądarce
5. **2014** SPIR — język pośredni shaderów
6. **2016** Vulkan — niskopoziomowy API

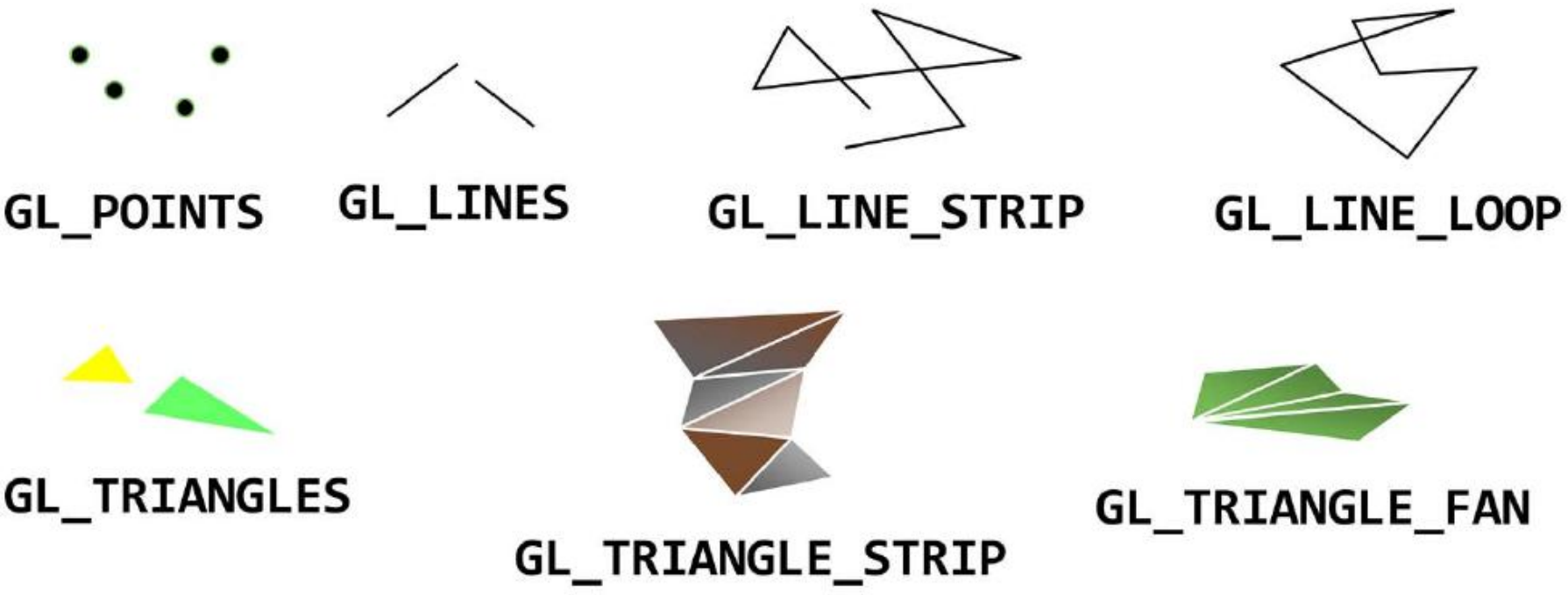
OpenGL: Khronos konsorcjum

nowości z prezentacji Khronos na konferencji SIGGRAPH 2017

OpenGL: API dla std. potoku grafiki 3D

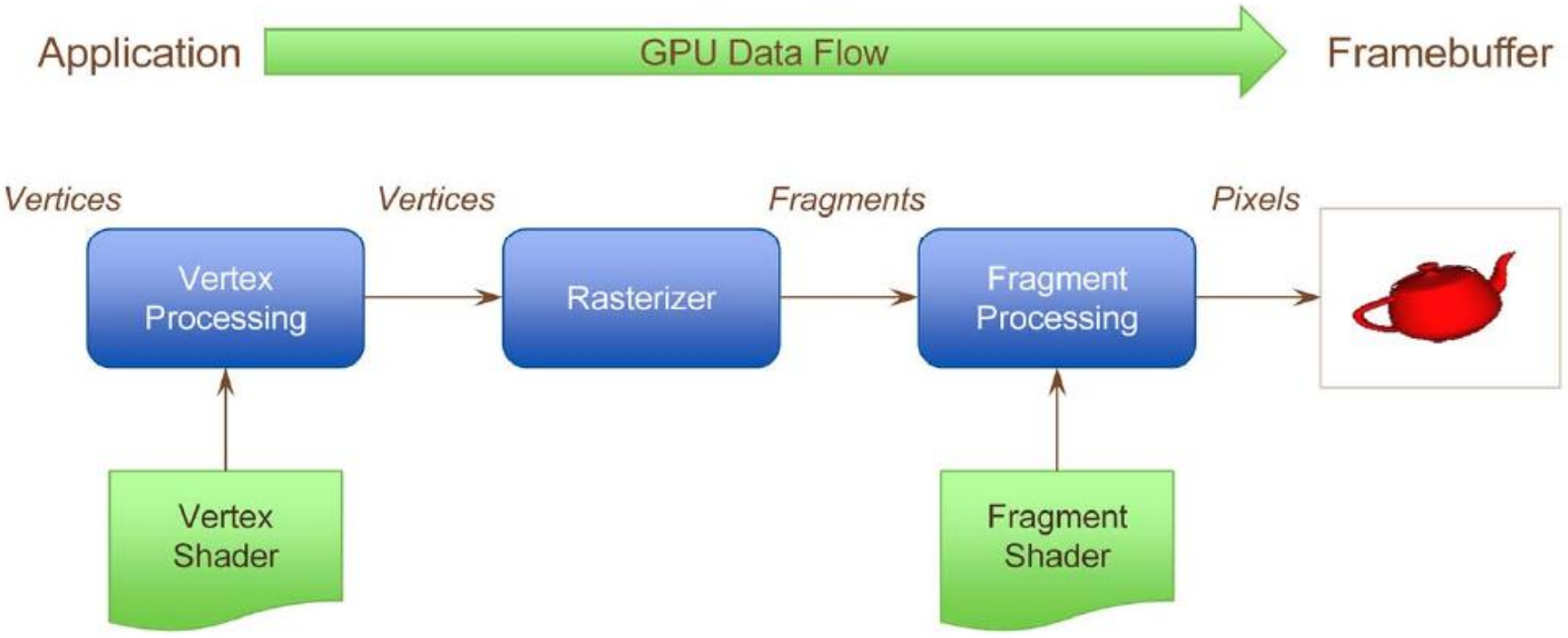
Obrazki na następnych slajdach z dokumentacji OpenGL i Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

- All primitives are specified by vertices



OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:



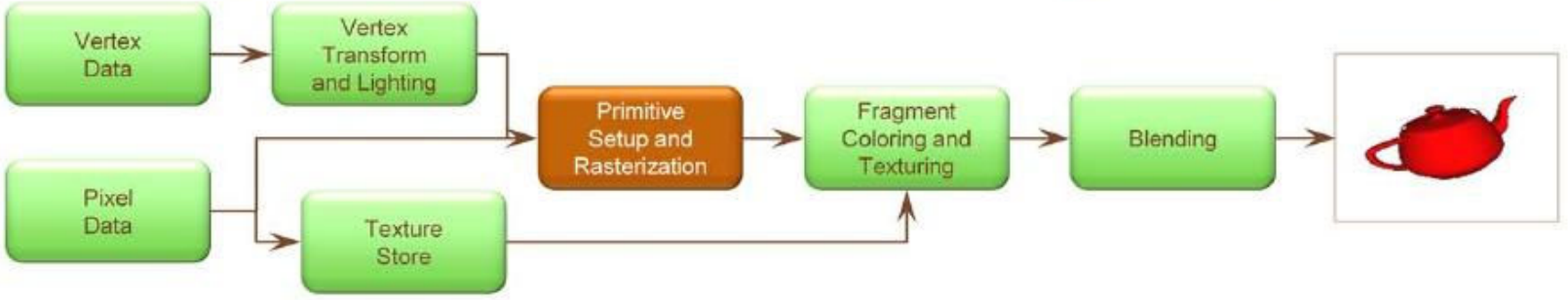
OpenGL versus DirectX

OpenGL: API dla std. potoku grafiki 3D

Std. potok grafiki razem z OpenGL API powstał w firmie SGI

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

- OpenGL 1.0 was released on July 1st, 1994 ~~2~~
- Its pipeline was entirely *fixed-function*
  - the only operations available were fixed by the implementation

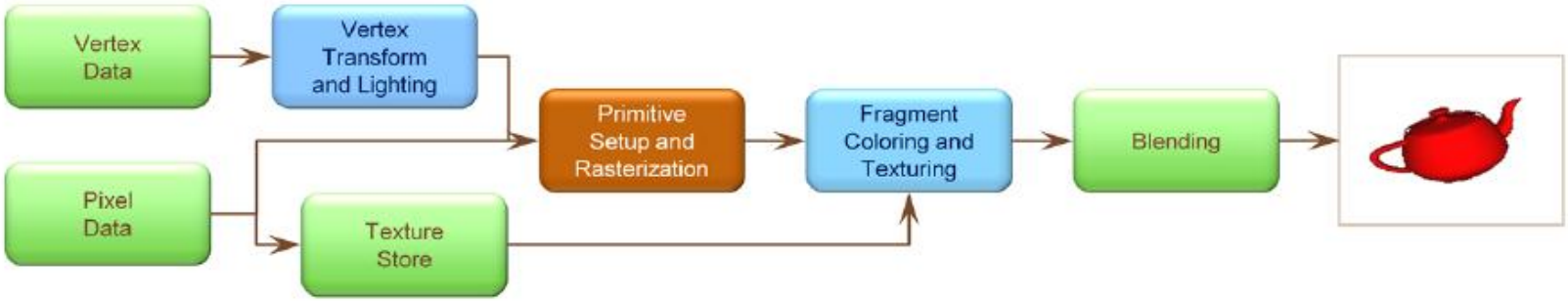


- The pipeline evolved, but remained fixed-function through OpenGL versions 1.1 through 2.0 (Sept. 2004)

OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

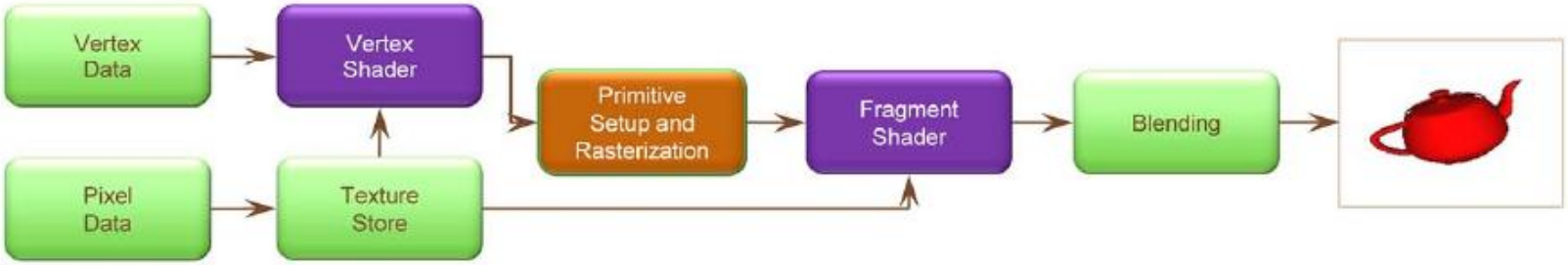
- OpenGL 2.0 (officially) added programmable shaders
  - *vertex shading* augmented the fixed-function transform and lighting stage
  - *fragment shading* augmented the fragment coloring stage
- However, the fixed-function pipeline was still available



OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

- OpenGL 3.1 removed the fixed-function pipeline
  - programs were required to use only shaders

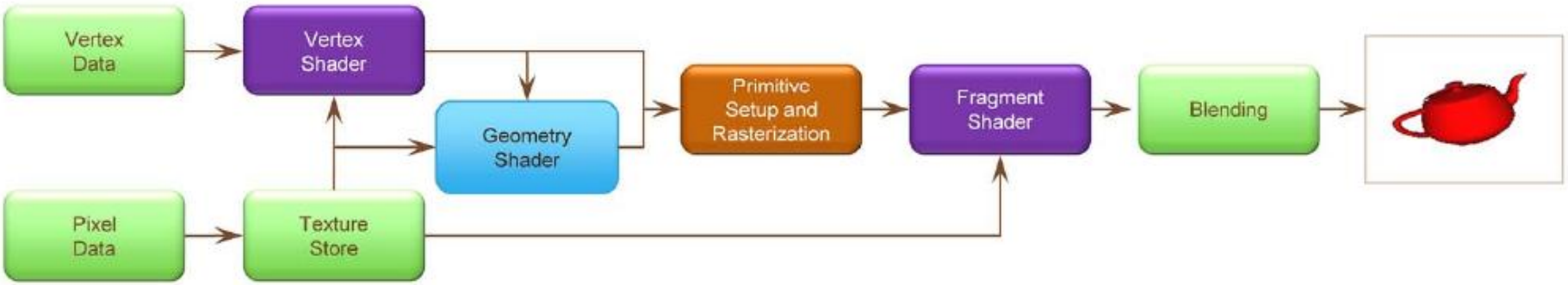


- Additionally, almost all data is *GPU-resident*
  - all vertex data sent using buffer objects

OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

- OpenGL 3.2 (released August 3rd, 2009) added an additional shading stage – *geometry shaders*



OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

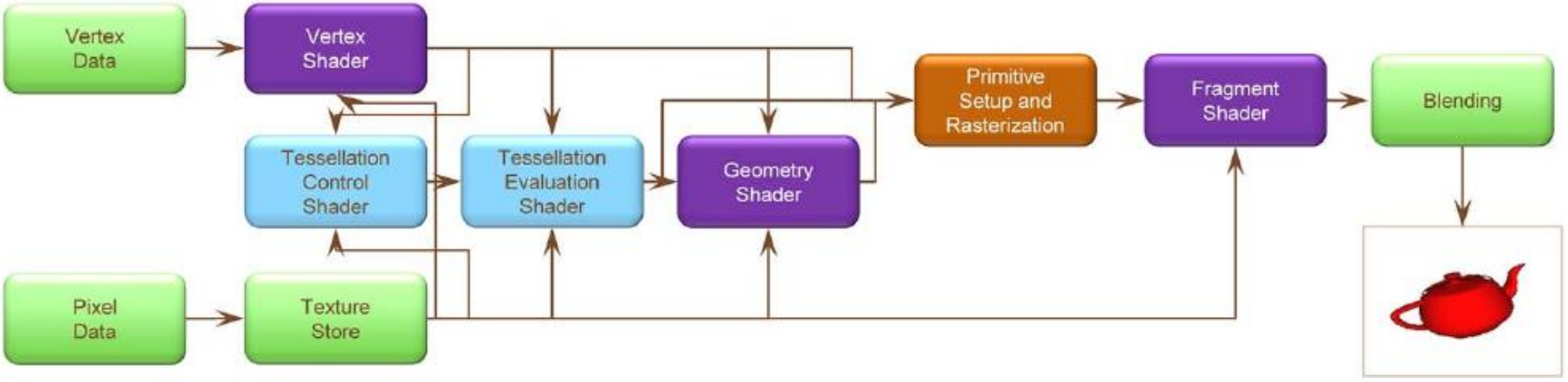
- OpenGL 3.2 also introduced *context profiles*
  - profiles control which features are exposed
    - it’s like `GL_ARB_compatibility`, only not insane ☺
  - currently two types of profiles: *core* and *compatible*

Context Type	Profile	Description
Full	core	All features of the current release
	compatible	All features ever in OpenGL
Forward Compatible	core	All non-deprecated features
	compatible	Not supported

OpenGL: API dla std. potoku grafiki 3D

Ed Angel, Dave Schreiner — “An Introduction to modern OpenGL Prog.”:

- OpenGL 4.1 (released July 25th, 2010) included additional shading stages – *tessellation-control and tessellation-evaluation shaders*



OpenGL: podstawowe cechy

Model i działanie:

- maszyna ze stanami
- model klient — serwer
- ustalamy stan potoku: zmienne, transformacje, programy shaderów, tworzymy bufory na dane
- przesyłamy dane dla wierzchołków z CPU na GPU
- *“DrawCall”* — żądanie rysowania dla danych na GPU

OpenGL: stare API (deprecated)

```
1  main() {
2     OpenWindow();
3
4     glClear(GL_COLOR_BUFFER_BIT);
5     glColor3f(1.0,1.0,1.0);
6     glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
7     glBegin(GL_POLYGON);
8        glVertex2f(-0.5, -0.5);
9        glVertex2f(-0.5,  0.5);
0        glVertex2f( 0.5,  0.5);
1        glVertex2f( 0.5, -0.5);
2     glEnd();
3     glFlush();
4
5     KeepWindowForAwhile();
6  }
```

OpenGL: nowa wersja API

W nowym OpenGL

- nie można używać trybu bezpośredniego glBegin/glEnd. dane (dowolne!) muszą być przygotowane w tablicach
- VAO (Vertex Array Object - stany tablicy wierzołków)
- VBO na karcie GPU (Vertex Buffer Object - dane wierz.)
- GLSL — język dla shaderów
- aby cokolwiek narysować konieczne są programy do przetwarzania wierzchołków i pikseli (vertex, fragment shader)

Po przygotowaniu danych rysowanie to np.:

```
1       glDrawArrays(GL_LINES, start, count);
```

OpenGL: inne biblioteki

- OpenGL nie obsługuje tworzenia okna oraz interakcji (klawiatura i mysz)
- Nieprzenośne biblioteki interfejsu z systemem okienkowym: GLW(Microsoft), GLX(X-Windows), AGL(Apple)
- Przenośne biblioteki zapewniające tworzenie okien, kontekstu i interakcje: GLUT, SDL, GLFW, ...
- Dodatkowe: GLEW (extension wrangler), GLM (math: wektory)
- Widgety: AntTweakBar, GLUI, nanoGui, imgui, ...

OpenGL: GLFW tutorial01

```
1  #include <stdio.h>     // Include standard headers
2  #include <stdlib.h>
3  #include <GL/glew.h>   // Include GLEW
4  #include <glfw3.h>     // Include GLFW
5
6  GLFWwindow* window;
7
8  #include <glm/glm.hpp> // Include GLM
9  using namespace glm;
0
1  int main( void ){
2          if( !glfwInit() ) {     // Initialise GLFW
3                  fprintf( stderr, "Failed to initialize GLFW\n" );
4                  getchar();
5                  return -1;
6          }
7          glfwWindowHint(GLFW_SAMPLES, 4);
8          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
9          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
0          glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // To make MacOS happ
1          glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

OpenGL: GLFW tutorial01

```
3         // Open a window and create its OpenGL context
4         window = glfwCreateWindow( 1024, 768, "Tutorial 01", NULL, NULL);
5         if( window == NULL ){
6                 fprintf( stderr, "Failed to open GLFW window. If you have an Intel
7                 getchar();
8                 glfwTerminate();
9                 return -1;
0         }
1         glfwMakeContextCurrent(window);
2
3         // Initialize GLEW
4         if (glewInit() != GLEW_OK) {
5                 fprintf(stderr, "Failed to initialize GLEW\n");
6                 getchar();
7                 glfwTerminate();
8                 return -1;
9         }
0
1         // Ensure we can capture the escape key being pressed below
2         glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
```

OpenGL: GLFW tutorial01

```
4         // Dark blue background
5         glClearColor(0.0f, 0.0f, 0.4f, 0.0f);
6
7         do{
8                 // Clear the screen. It's not mentioned before Tutorial 02, but it
9                 glClear( GL_COLOR_BUFFER_BIT );
0
1                 // Draw nothing, see you in tutorial 2 !
2
3                 // Swap buffers
4                 glfwSwapBuffers(window);
5                 glfwPollEvents();
6
7         } // Check if the ESC key was pressed or the window was closed
8         while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS &&
9                    glfwWindowShouldClose(window) == 0 );
0         // Close OpenGL window and terminate GLFW
1         glfwTerminate();
2         return 0;
3 }
```

OpenGL: shadery w GLSL

```
4       const GLchar* vs[] = {
5          "#version 330 core\n",
6          //      "layout(location = 0) in vec2 pos;",
7          "in vec2 pos;",
8          "out vec2 uv;",
9          "void main(void) {",
0          "   gl_Position = vec4(pos, 0.0, 1.0); ",
1          "   uv = 0.5*pos + 0.5;",
2          "}"
3       };
4       const GLchar* fs[] = {
5          "#version 130\n",
6
7          "in  vec2 uv;",
8          "out vec4 color;",
9          "void main(void) {",
0          "   color = vec4(uv.x,uv.y,0.,1.);",
1          "} "
2       };
```

OpenGL: użycie shaderów

```
3       GLuint v = glCreateShader(GL_VERTEX_SHADER);
4       GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
5       glShaderSource(v, 7, vs, NULL);
```

... ...

```
0
1       CompileShaders(v,f);  // Auxilary function
2       glUseProgram(p());
3
4       a0 = glGetAttribLocation(p(),  "pos");
```

Numer atrybutu (dla uniform podobnie) można uzyskać po nazwie j.w. lub wymusić w GLSL przez:

```
1 layout(location = 0) in vec3 pos;
```

OpenGL: przygotowanie danych

```
10  void setBuffers() {
11     glGenVertexArrays(1, &vaoId);
12     glGenBuffers(     1, &vboId);
13     glBindVertexArray(           vaoId);
14     glBindBuffer(GL_ARRAY_BUFFER, vboId);
15
16     GLfloat vert[][2] = {  // TRI FAN
17        { -.9,  -.9 },
18        { .9, -.9 },
19        { .9,  .9 },
20        { -.9,  .9 }     };
21     glBufferData(GL_ARRAY_BUFFER, 4*2*sizeof(float),
22                  vert, GL_STATIC_DRAW );
23     glEnableVertexAttribArray(a0);
24     glVertexAttribPointer(
25        a0,                  // attribute 0,
26        2,                   // size
27        GL_FLOAT,            // type
28        GL_FALSE,            // normalized?
29        0,//24,              // stride
30        (void*)0             // array buffer offset
31     );
```

OpenGL: rysowanie

```
3    void draw() {
4       glUseProgram(p());
5       glBindVertexArray(           vaoId);
6       glBindBuffer(GL_ARRAY_BUFFER,vboId);
7       glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
8    }
```

OpenGL: przykłady

logo + geometry shader

www.shadertoy.com

offline shadertoy

...