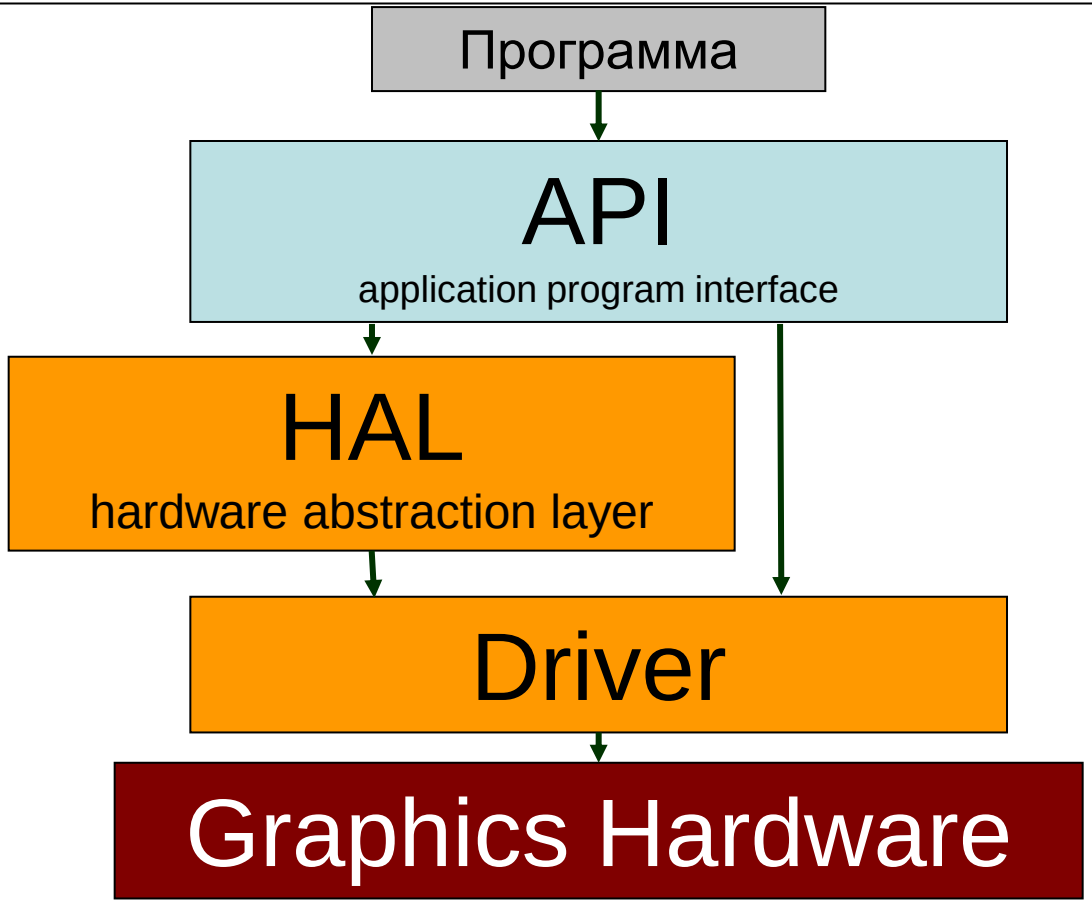
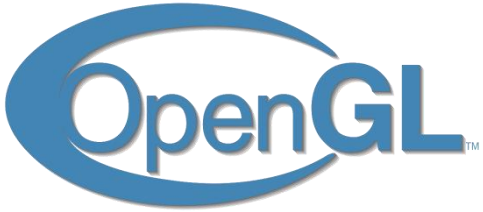
Графика реального времени. OpenGL. Часть 1

материалы занятий: https://compsciclub.ru/courses/graphics2018/2018-autumn/classes/
дублируются на сайте: http://www.school30.spb.ru/cgsg/cgc2018/

Структура взаимодействия приложения с аппаратурой





- **Open GL** – open graphics library
(SGI, 90-е годы, версии 1.0-4.2)
  - GLUT - OpenGL Utility Toolkit (http://www.opengl.org/resources/libraries/glut/)
  - FreeGLUT – Free OpenGL Utility Toolkit (http://sourceforge.net/projects/freeglut/)
  - GLFW - Graphics Library Framework (http://www.glfw.org)
  - GLEW - OpenGL Extension Wrangler Library (http://glew.sourceforge.net/)
  - GLM - OpenGL Mathematics (http://glm.g-truc.net/)

- **Microsoft DirectX Graphics** – часть MS DirectX API (Direct3D)
(1995 Microsoft Corp., Windows Game SDK, версии 1.0-12.0)
  - D3DX – retained mode toolkit
  - XNA - Xbox New Architecture (http://msdn.microsoft.com/ru-ru/xna/)
  - Microsoft DXR (DirectX Raytracing)

- **Vulkan** – кроссплатформенны API для 2D и 3D графики
(2016 Khronos Group, версии 1.0-1.1.95)
  - Vulkan API (http:// www.khronos.org/vulkan/)
- **Metal** – Apple Inc. (2014, macOS, iOS,)

Модель анимации

- Анимация (основной цикл программы)
  - Взаимодействие с ОС
  - Синхронизация по времени
  - Опрос устройств ввода
- Вывод (рендеринг)
  - Инициализация API
  - Вывод кадра
  - Хранение геометрических объектов

Синхронизация по времени

Old style: *(классический вариант со стандартной библиотекой)*

```
  #include <time.h>
  clock() -> tick -> CLOCKS_PER_SEC
```

на старте программы:

```
  clock_t StartTime = clock();
  . . .
```

на каждом шаге:

```
  clock_t t = clock();
  GlobalTimeInSec = (t - StartTime) / (double)CLOCKS_PER_SEC;
```

Синхронизация по времени

High resolution timer: *(вариант с таймеров высокого разрешения - WinAPI)*

```
  unsigned long long
    StartTime,  /* Start program time */
    TimePerSec; /* Timer resolution */
```

на старте программы:

```
  LARGE_INTEGER t;
```

определить точность таймера (тики в секунду)

```
  QueryPerformanceFrequency(&t);
  TimePerSec = t.QuadPart;
```

определить время начала

```
  QueryPerformanceCounter(&t);
  StartTime = t.QuadPart;
```

на каждом шаге:

```
  LARGE_INTEGER t;
  QueryPerformanceCounter(&t);
  GlobalTimeInSec = (double)(t.QuadPart - StartTime) / TimePerSec;
```

Синхронизация по времени

```
-- время начала программы
     StartTime
-- общее время работы - global time
     t - StartTime
-- межкадровая задержка времени (без учета паузы) - global delta time
     t - OldTime    (OldTime – время прошлого кадра)
-- pause flag:
     IsPause
-- локальное время (с учетом паузы) - local time
     t - PauseTime - StartTime
-- межкадровая задержка с учетом паузы - local delta time
     == global delta time  если !IsPause
     == 0                  если IsPause
для вычисления PauseTime:
  если IsPause:
    PauseTime += t - OldTime
-- для определения количества кадров в секунду - frames per second (FPS):
     1 time per second:
       FrameCounter – счетчик кадров     OldTimeFPS   – время прошлого замера FPS
```

Синхронизация по времени

```
LARGE_INTEGER t;
QueryPerformanceCounter(&t);
. . .
/* FPS */
FrameCounter++;
if (t.QuadPart - OldTimeFPS > TimePerSec)
{
  FPS = FrameCounter * TimePerSec / (DBL)(t.QuadPart - OldTimeFPS);
  OldTimeFPS = t.QuadPart;
  FrameCounter = 0;
}
```

Файлы заголовков:

```
#include <gl/gl.h>
#include <gl/glu.h>
```

Библиотеки:

```
opengl32 + glu32
```

Об именах в GL:

Сокращения: **GL** - **OpenGL** **GLU** — *вспомогательные функции*
GLUT — *интерфейсная часть и т.п. от GLUT*
GLEW — *библиотека работы с расширениями*

префиксы:

glNameName — функции: **glClear(0); glPolygonMode(...);**
GL_NAME_NAME — константы: **GL_FRONT_AND_BACK, GL_POINTS**
GLname — внутренние типы: **GLbyte**

OpenGL+GLEW

OpenGL Extension Wrangler Library

Подключение расширений – дополнений и новых функций (необходим всегда для современных версий, подключает все новые возможности OpenGL, беря на себя нужные инициализации)
http://glew.sourceforge.net

Подключение к проекту:

```
#define GLEW_STATIC   <-- отказ от glew32.dll
#include <glew.h>     <-- подключается до GL/***.H
   + библиотека: glew32s
```

Graphics Library Utility Toolkit

GLUT (~1997)

http://www.opengl.org/resources/libraries/glut/glut_downloads.php

FreeGLUT (~2011)

http://sourceforge.net/projects/freeglut/

Обработка всего, что связанно с интерфейсом: окна, создание "цикла" сообщений и т.п., организация ввода (мышь, клавиатура, джойстик и т.д.).
События обрабатываются с помощью функций "обратного вызова" (CALLBACK).

Подключение

#include "glut.h"

+ библиотека **glut32** (обычно подключается через **glut.h**)

В OS Windows необходима динамическая библиотка **glut32.dll**

GLUT

```
#include <stdlib.h>
#include <glut.h>

void Display( void )
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  . . .
  glFinish();
  glutSwapBuffers();
  glutPostRedisplay();
}

void Keyboard( unsigned char Key, int X, int Y )
{
  if (Key == 27)
    exit(0);
}
```

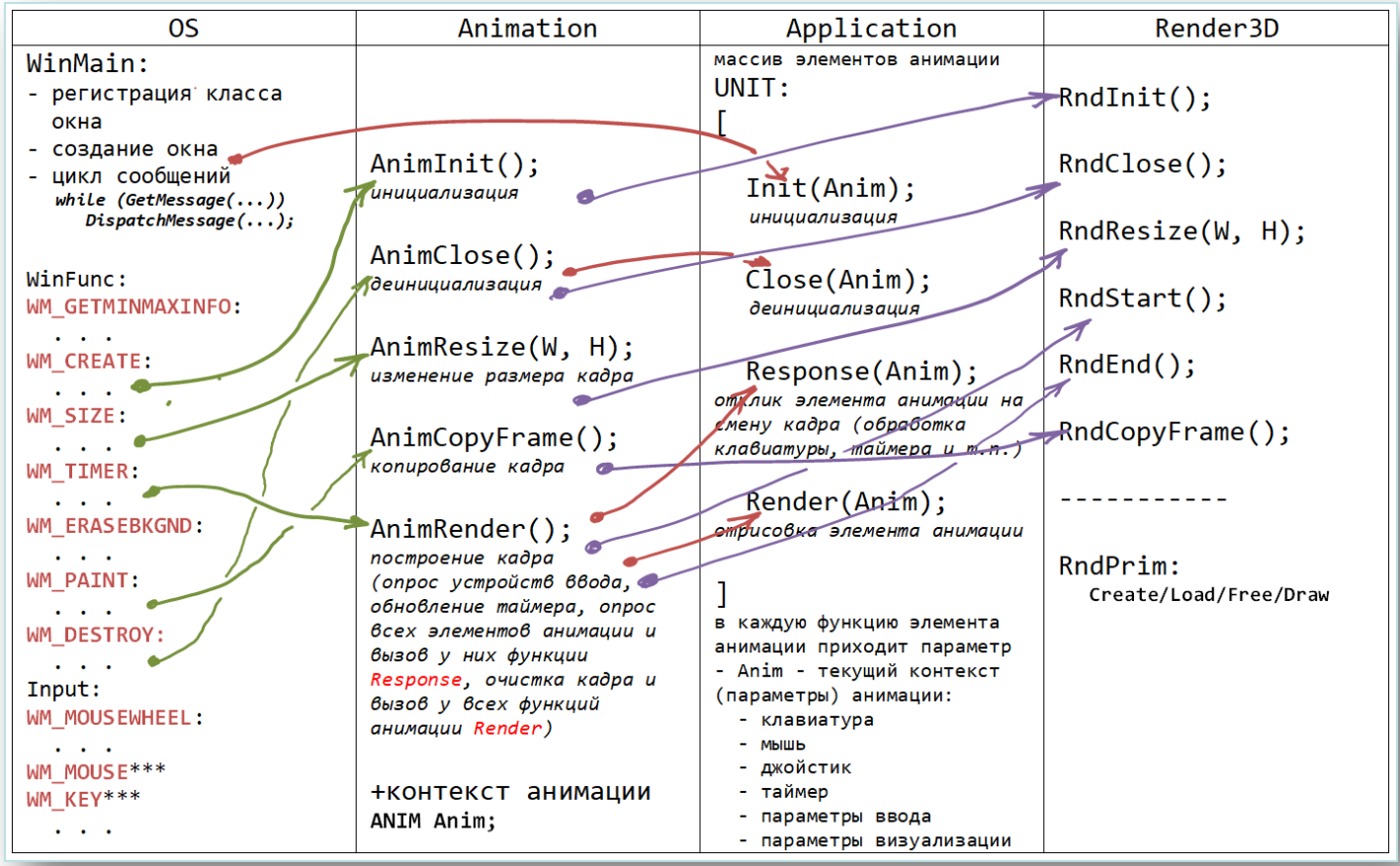
```
int main( int argc, char *argv[] )
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Sample");

  glutDisplayFunc(Display);
  glutKeyboardFunc(Keyboard);

  glutMainLoop();
  return 0;
}
```

WinAPI

```
#include <windows.h>

#define WND_CLASS_NAME "My window class"

/* Главная функция программы */
INT WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    CHAR *CmdLine, INT ShowCmd )
{
  WNDCLASS wc;
  HWND hWnd;
  MSG msg;

  wc.style = CS_VREDRAW | CS_HREDRAW;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.lpszMenuName = NULL;
  wc.hInstance = hInstance;
  wc.lpfnWndProc = MyWindowFunc;
  wc.lpszClassName = WND_CLASS_NAME;

  /* Регистрация класса в системе */
  if (!RegisterClass(&wc))
  {
    MessageBox(NULL, "Error register window class", "ERROR", MB_OK);
    return 0;
  }
```

```
  /* Создание окна */
  hWnd =
    CreateWindow(WND_CLASS_NAME, "Title",
      WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, CW_USEDEFAULT,
      CW_USEDEFAULT, CW_USEDEFAULT,
      NULL, NULL, hInstance, NULL);
  /* Показать и перерисовать окно */
  ShowWindow(hWnd, SW_SHOWNORMAL);
  UpdateWindow(hWnd);
  /* Цикл обработки сообщений */
  while (TRUE)
  {
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
      if (msg.message == WM_QUIT)
        break;
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    else
      . . .
  }
  return msg.wParam;
} /* End of 'WinMain' function */
```

```
/* Функция обработки сообщения окна */
LRESULT CALLBACK MyWindowFunc( HWND hWnd, UINT Msg,
                               WPARAM wParam, LPARAM lParam )
{
  HDC hDC;
  PAINTSTRUCT ps;
  int w, h;

  switch (Msg)
  {
  case WM_CREATE:
    SetTimer(hWnd, 30, 3, NULL);
    return 0;
  case WM_SIZE:
    w = LOWORD(lParam);
    h = HIWORD(lParam);
    return 0;
  case WM_ERASEBKGND:
    return 1;
  case WM_TIMER:
    return 0;
  case WM_PAINT:
    hDC = BeginPaint(hWnd, &ps);
    EndPaint(hWnd, &ps);
    return 0;
  case WM_DESTROY:
    KillTimer(hWnd, 30);
    PostQuitMessage(0);
    return 0;
  }
  return DefWindowProc(hWnd, Msg, wParam, lParam);
} /* End of 'MyWindowFunc' function */
```


OS
Animation
Application
Render3D
WinMain:
- регистрация класса окна
- создание окна
- цикл сообщений
while (GetMessage(...))
DispatchMessage(...);
WinFunc:
WM_GETMINMAXINFO:
. . .
WM_CREATE:
. . .
WM_SIZE:
. . .
WM_TIMER:
. . .
WM_ERASEBKGND:
. . .
WM_PAINT:
. . .
WM_DESTROY:
. . .
Input:
WM_MOUSEWHEEL:
. . .
WM_MOUSE***
WM_KEY***
. . .
AnimInit();
инициализация
AnimClose();
деинициализация
AnimResize(W, H);
изменение размера кадра
AnimCopyFrame();
копирование кадра
AnimRender();
построение кадра (опрос устройств ввода, обновление таймера, опрос всех элементов анимации и вызов у них функции Response, очистка кадра и вызов у всех функций анимации Render)
+контекст анимации
ANIM Anim;
массив элементов анимации
UNIT:
[
Init(Anim);
инициализация
Close(Anim);
деинициализация
Response(Anim);
отклик элемента анимации на смену кадра (обработка клавиатуры, таймера и т.п.)
Render(Anim);
отрисовка элемента анимации
]
в каждую функцию элемента анимации приходит параметр
- Anim - текущий контекст (параметры) анимации:
- клавиатура
- мышь
- джойстик
- таймер
- параметры ввода
- параметры визуализации
RndInit();
RndClose();
RndResize(W, H);
RndStart();
RndEnd();
RndCopyFrame();

RndPrim:
Create/Load/Free/Draw

WinAPI: инициализация OpenGL

```
/* Windows specific global data */
HWND  VG4_hRndWnd;
HDC VG4_hRndDC;
HGLRC VG4_hRndGLRC;

RndInit:
  INT i;
  PIXELFORMATDESCRIPTOR pfd = {0};

  /* Store window and create device context */
  VG4_hRndWnd = hWnd;
  VG4_hRndDC = GetDC(hWnd);
  /* OpenGL init: pixel format setup */
  pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);
  pfd.nVersion = 1;
  pfd.dwFlags = PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL;
  pfd.cColorBits = 32;
  pfd.cDepthBits = 32;
  i = ChoosePixelFormat(VG4_hRndDC, &pfd);
  DescribePixelFormat(VG4_hRndDC, i, sizeof(pfd), &pfd);
  SetPixelFormat(VG4_hRndDC, i, &pfd);

  /* OpenGL init: setup rendering context */
  VG4_hRndGLRC = wglCreateContext(VG4_hRndDC);
  wglMakeCurrent(VG4_hRndDC, VG4_hRndGLRC);
```

```
  /* GLEW initialize */
  if (glewInit() != GLEW_OK ||
      !(GLEW_ARB_vertex_shader && GLEW_ARB_fragment_shader))
  {
    wglMakeCurrent(NULL, NULL);
    wglDeleteContext(VG4_hRndGLRC);
    ReleaseDC(VG4_hRndWnd, VG4_hRndDC);
    exit(0);
  }
  /* OpenGL set render parameters */
  glClearColor(0.3, 0.5, 0.7, 1);

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

  glEnable(GL_PRIMITIVE_RESTART);
  glPrimitiveRestartIndex(-1);

RndClose:
  wglMakeCurrent(NULL, NULL);
  wglDeleteContext(VG4_hRndGLRC);
  ReleaseDC(VG4_hRndWnd, VG4_hRndDC);
```

Обработка вывода OpenGL

```
/*** Rendering specific data ***/
/* Projection data */
float
  VG4_RndProjSize = 0.1,     /* Project plane unit square size */
  VG4_RndProjDist = 0.1,     /* Distance to project plane (near) */
  VG4_RndProjFarClip = 1000; /* Distance to project far clip plane (far) */
int
  VG4_RndFrameW, VG4_RndFrameH; /* Viewport size */
/* Transformation matrix /*/
MATR
  VG4_RndViewMatr, /* View coordinate system matrix */
  VG4_RndProjMatr; /* Projection matrix */
/* Camera parameters */
VEC
  VG4_RndCamLoc,   /* Camera location */
  VG4_RndCamRight, /* Camera right direction */
  VG4_RndCamUp,    /* Camera up direction */
  VG4_RndCamDir;   /* Camera forward direction */

RndResize:
  float ratio_x = 1, ratio_y = 1;

  if (VG4_RndFrameW >= VG4_RndFrameH)
    ratio_x *= (float)VG4_RndFrameW / VG4_RndFrameH;
  else
    ratio_y *= (float)VG4_RndFrameH / VG4_RndFrameW;

  VG4_RndProjMatr =
    MatrFrustum(-ratio_x * VG4_RndProjSize / 2, ratio_x * VG4_RndProjSize / 2,
                -ratio_y * VG4_RndProjSize / 2, ratio_y * VG4_RndProjSize / 2,
                VG4_RndProjDist, VG4_RndProjFarClip);
  glViewport(0, 0, W, H);
```

```
RndCopyFrame:
  SwapBuffers(VG4_hRndDC);

RndStart:
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

RndEnd:
  glFinish();

AnimRender:
  // опрос устройств ввода
  // опрос таймера
  // вызов у объектов анимации Response
  VG4_RndStart();
  // вызов у объектов анимации Render
  VG4_RndEnd();
```

OpenGL: Устаревший вывод

```
glBegin(режим задания примитивов);
  ...
  команды задания примитивов
glEnd();
```

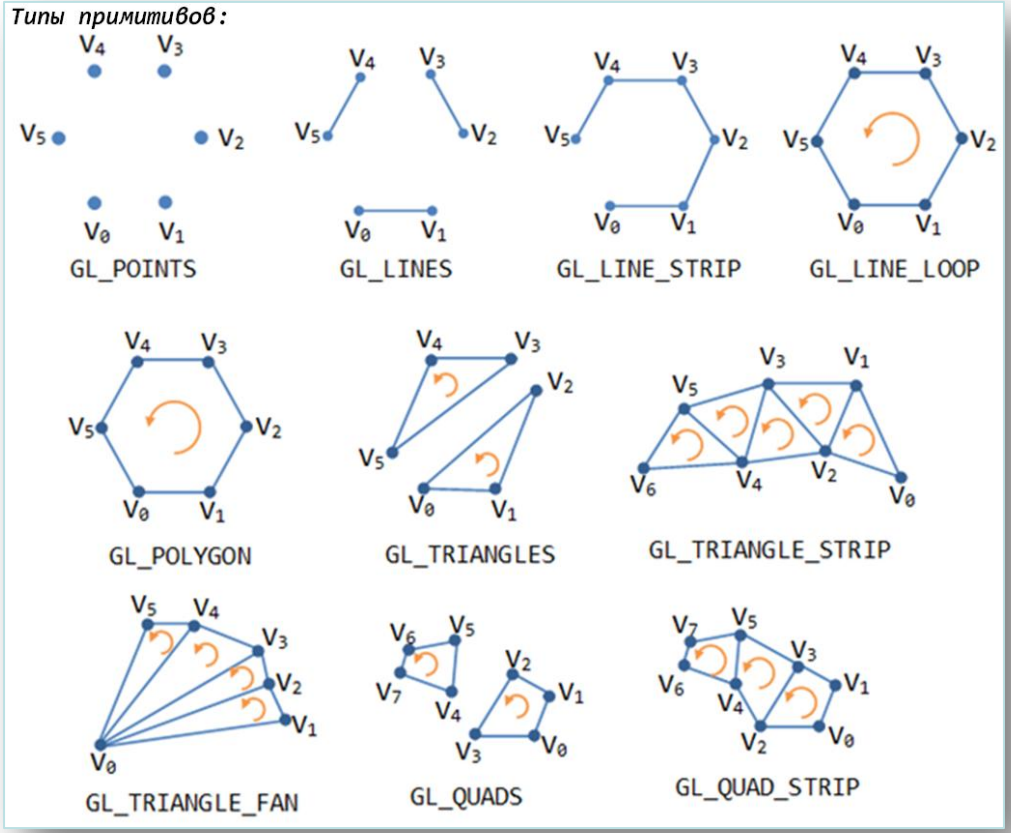
Например:

```
glBegin(GL_TRIANGLES);
  glColor3d(1, 1, 0);
  glVertex3d(0.15, 0, 0.15);
  glVertex3d(0.15, 0, 8.8);
  glVertex3d(8.8, 0, 0.15);
glEnd();
```

Например, рисование осей X, Y и Z:

```
glBegin(GL_LINES);
  glColor3d(1, 0, 0);
  glVertex3d(0, 0, 0);
  glVertex4d(1, 0, 0, 0);
  glColor3d(0, 1, 0);
  glVertex3d(0, 0, 0);
  glVertex4d(0, 1, 0, 0);
  glColor3d(0, 0, 1);
  glVertex3d(0, 0, 0);
  glVertex4d(0, 0, 1, 0);
glEnd();
```

Примитивы OpenGL



Примитивы OpenGL

-- Точечные примитивы

- `GL_POINTS` — отрисовка отдельных точек (каждая вершина - отдельная точка)

Дополнительные параметры точечных объектов:

- Размер точки
  `glPointSize(`*размер*`);`
- Сглаживание для точек (круглые окончания)
  `glEnable(GL_POINT_SMOOTH);` — включение (выключение — `glDisable(GL_POINT_SMOOTH);`)

-- Линейные примитивы

- `GL_LINES` — отрисовка отдельных отрезков (каждая пара вершин - отрезок)

Дополнительные параметры линейных объектов:

- Ширина линии
  `glLineWidth(`ширина`);`
- Сглаживание линий
  `glEnable(GL_LINE_SMOOTH);`
- Интерполяция цвета между вершинами
  `glShadeModel(GL_FLAT` или `GL_SMOOTH);` — включение (`GL_SMOOTH` — включен по умолчанию)
- Шаблон
  `glEnable(GL_LINE_STIPPLE);` — включение отрисовки линии по шаблону задание шаблона:
  `glLineStipple(`*множитель*`,` *16-ти битный шаблон*`);`

- `GL_LINE_STRIP` — отрисовка ломаной
- `GL_LINE_LOOP` — отрисовка замкнутой ломаной

Примитивы OpenGL

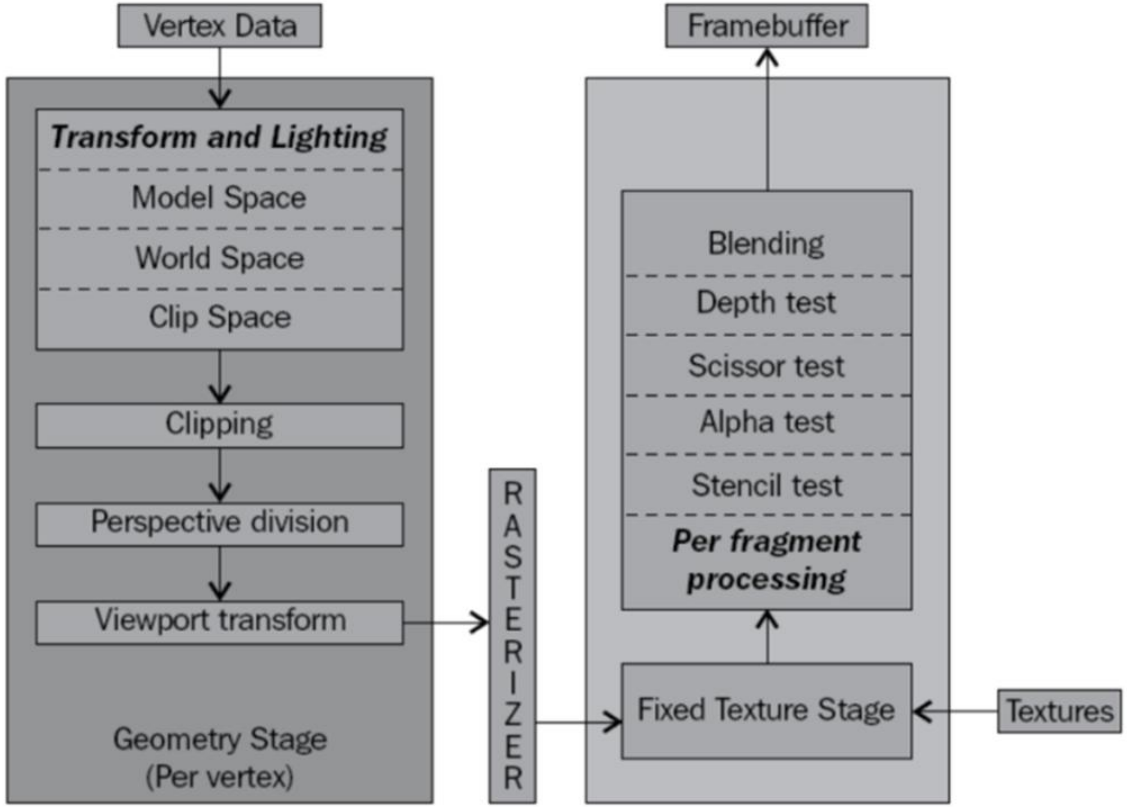
-- Площадные примитивы

- `GL_TRIANGLES` — отрисовка треугольника (каждая тройка вершин)
- `GL_TRIANGLE_STRIP` — полоса треугольников
- `GL_TRIANGLE_FAN` — веер треугольников
- `GL_QUADS` — каждые 4 вершины - четырехугольник (выпуклый) [*устаревший*]
- `GL_QUAD_STRIP` — полоса четырехугольников [*устаревший*]
- `GL_POLYGON` — выпуклый многоугольник [*устаревший*]

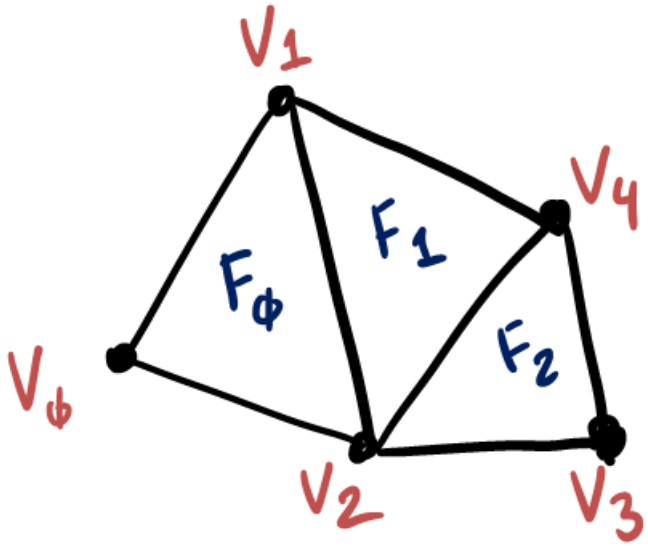
Дополнительные параметры площадных объектов:

- У площадных фигур определено понятие лицевой и тыльной стороны. Определяется это по обходу вершин — по умолчанию обходя вершины против часовой стрелки считается, что смотрим на лицевую сторону.
  `glFrontFace(GL_CW или GL_CCW);` — определяет какая сторона лицевая
  (`GL_CW` — *clock-wise* — по часовой стрелке, `GL_CCW` — *counter-clock-wise* — против — по умолчанию)
- Отмена рисования определенных сторон (*culling*)
  `glEnable(GL_CULL_FACE);` — включение “нерисования” определенных сторон (по умолчанию — тыльных)
  кого не рисовать:
  `glCullFace(GL_BACK или GL_FRONT или GL_FRONT_AND_BACK);`
- Сглаживание (устранение ступенчатости)
  `glEnable(GL_POLYGON_SMOOTH);` — сглаживание
- Интерполяция цвета (закраска по *Гуро* [*Gouraud shading*])
  `glShadeModel` (режим интерполяции цвета — `GL_FLAT` без сглаживания, `GL_SMOOTH` — со сглаживанием);
- Использование трафарета
  `glEnable(GL_POLYGON_STIPPLE);` — отрисовка многоугольников по трафарету:
  `glPolygonStipple` (*маска трафарета*); — задает маску — битовый массив 32 на 32 точки (`long m[32];`)
- Общие параметры построения многоугольников:
  `glPolygonMode` (*кого*, *как*);
  кого – `GL_BACK` или `GL_FRONT` или `GL_FRONT_AND_BACK`
  как – `GL_FILL` или `GL_LINE` или `GL_POINT`

Render


Vertex Data
Transform and Lighting
Model Space
World Space
Clip Space
Clipping
Perspective division
Viewport transform
Geometry Stage
(Per vertex)
RASTERIZER
Framebuffer
Blending
Depth test
Scissor test
Alpha test
Stencil test
Per fragment processing
Fixed Texture Stage
Textures

Топология



$$V = \{V_0, V_1, V_2, V_3, V_4\}$$
$$I = \{0, 2, 1, 2, 4, 1, 2, 3, 4\}$$

Vertex Buffer Object (VBO)

VBO – vertex buffer object (примитив в видеопамяти):

для работы необходимы - массив вершин и буфер вершин:
vertex array (описание данных) – кто где
vertex buffer (сами данные) – VERTEX

массив вершин - ***VertexArray*** –
связка посылаемых данных сверху вниз (***layout***).

буфер вешин - ***VertexBuffer*** –
массив данных, отсылаемых в видеокарту.

Vertex Buffer Object (VBO)

Хранение вершин:

```
/* Структура хранения данных о вершине */
typedef struct tagvg4VERTEX
{
  VEC P;    /* позиция */
  VEC2 T;   /* текстурные координаты */
  VEC N;    /* нормаль */
  VEC4 C;   /* Цвет */
} vg4VERTEX;

typedef struct tagVEC2
{
  float X, Y;    /* координаты */
} VEC2;
```

```
typedef struct tagVEC
{
  float X, Y, Z;    /* координаты */
} VEC;

typedef struct tagVEC4
{
  float X, Y, Z, W;    /* координаты */
} VEC4;
```

Vertex Buffer Object (VBO)

Инициализационный этап

```
int VA, VBuf;

glGenBuffers(1, &VBuf);
glGenVertexArrays(1, &VA);

/* делаем активным массив вершин */
glBindVertexArray(VA);
```

заносим данные в буфер вершин:

```
/* делаем активным буфер */
glBindBuffer(GL_ARRAY_BUFFER, VBuf);
/* передаем данные (NumOfV - количество вершин, V - массив вершин) */
glBufferData(GL_ARRAY_BUFFER, sizeof(vg4VERTEX) * NumOfV, V, GL_STATIC_DRAW);
```

Vertex Buffer Object (VBO)

указываем в массиве вершин буфер и какие данные содержит:

```
/* присоединяем к массиву вершин буфер с данными (если еще не делали) */
glBindBuffer(GL_ARRAY_BUFFER, VBuf);

/* задаем порядок данных */
/*                    layout (номер атрибута),
 *                       количество компонент,
 *                          тип,
 *                                  надо ли нормировать,
 *                                      размер в байтах одного элемента буфера (stride),
 *                                                  смещение в байтах до начала данных */
glVertexAttribPointer(0, 3, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)0); /* позиция */
glVertexAttribPointer(1, 2, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)sizeof(VEC)); /* текстурные координаты */
glVertexAttribPointer(2, 3, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)(sizeof(VEC) + sizeof(VEC2))); /* нормаль */
glVertexAttribPointer(3, 4, GL_FLOAT, FALSE, sizeof(vg4VERTEX),
                      (VOID *)sizeof(VEC) * 2 + sizeof(VEC2)); /* цвет */
```

Vertex Buffer Object (VBO)

```
/* включаем нужные аттрибуты (layout) */
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);
glEnableVertexAttribArray(3);

/* выключили массив вершин */
glBindVertexArray(0);
```

удаление

```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* "отцепляем" буфер */
glBindBuffer(GL_ARRAY_BUFFER, 0);
glDeleteBuffers(1, &VBuf);
/* делаем неактивным массив вершин */
glBindVertexArray(0);
glDeleteVertexArrays(1, &VA);
```

Vertex Buffer Object (VBO)

Отрисовка:

```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* отрисовка */
glDrawArrays(GL_TRIANGLES, 0, NumOfV);
/* выключили массив вершин */
glBindVertexArray(0);
```

Vertex Buffer Object (VBO)

Индексы:

инициализационный этап

```
int IBuf;
. . .
glGenBuffers(1, &IBuf);
```

заносим данные в буфер индексов:

```
/* делаем активным буфер */
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBuf);
/* передаем данные (NumOfI - количество индексов, I - массив индексов) */
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(INT) * NumOfI, I, GL_STATIC_DRAW);
```

удаление

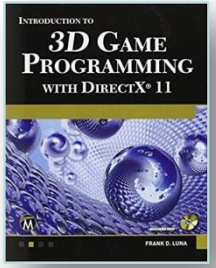
```
glDeleteBuffers(1, &IBuf);
```

Vertex Buffer Object (VBO)

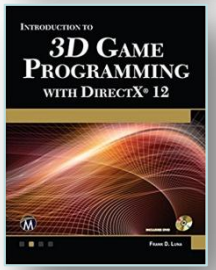
Отрисовка (индексированные примитивы):

```
/* делаем активным массив вершин */
glBindVertexArray(VA);
/* делаем активным массив индексов */
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBuf);
/* отрисовка */
glDrawElements(GL_TRIANGLES, NumOfI, GL_UNSIGNED_INT, NULL);
/* выключили индексы */
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
/* выключили массив вершин */
glBindVertexArray(0);
```

Литература



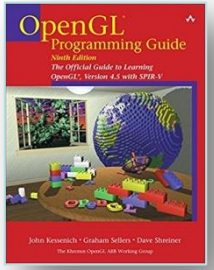
Frank Luna, ***«Introduction to 3D Game Programming with DirectX 11»***, Mercury Learning & Information, 2012.



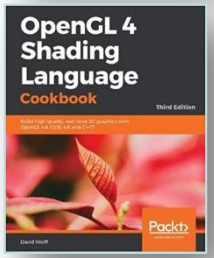
Frank Luna, ***«Introduction to 3D Game Programming with DirectX 12»***, Mercury Learning & Information, 2016.

Грэхем Селлерс, ***«Vulkan. Руководство разработчика»***, ДМК Пресс, 2017.

Литература



John Kessenich, Graham Sellers, Dave Shreiner,
«OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition)»,
Addison-Wesley Professional, 2016.



David Wolff, ***«OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++17, 3rd Edition»***,
Packt Publishing, 2018.

Дэвид Вольф, ***«OpenGL 4. Язык шейдеров. Книга рецептов»***,
ДМК Пресс, 2015.