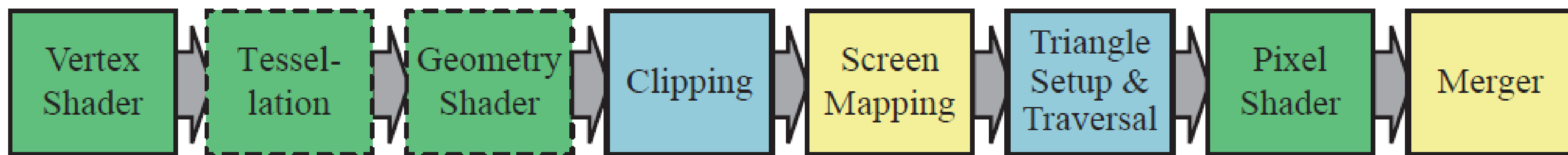
Графика реального времени. OpenGL. Часть 3.

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

Геометрический шейдер




```
Vertex data ->
  Vertex shader ->
    Tessellation control shader ->
      Tessellation evaluation shader ->
        Geometry shader ->
          Rasterizer (assembling primitives, rasterization) ->
            Fragment shader ->
              Raster operation (stencil, alpha, scissor, depth, blending) ->
                Framebuffer
```

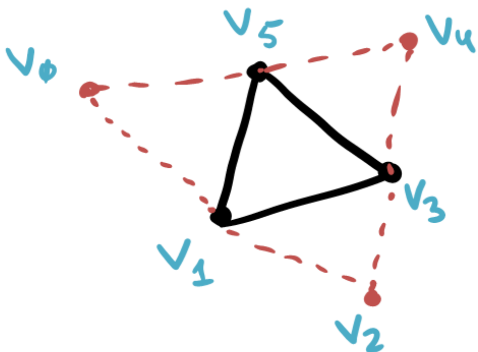
```
Vertex shader -> "gl_Position" -> Geometry shader -> {"gl_Position"}+primitive_type ->...
```

Геометрический шейдер

На вход – примитивы:

points *(GL_POINTS)*
lines *(GL_LINES, GL_LINE_STRIP, GL_LINES_LOOP)*
lines_adjancency *(GL_LINES_ADJANCENCY, GL_LINE_STRIP_ADJANCENCY)*

triangles *(GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN)*
triangles_adjancency *(GL_TRIANGLES_ADJANCENCY, GL_TRIANGLE_STRIP_ADJANCENCY)*



На выход – примитивы:

points
line_strip
triangle_strip

Геометрический шейдер

Структура описания 1-й вершины (в глобальном массиве **gl_in***):*

```
in gl_PerVertex
{
  vec4 gl_Position;  -- с вершинного (тесселляционного) шейдера
  . . .
} gl_in[];
```

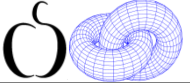
на выходе: **out vec4 gl_Position;**

порождение вершины: **EmitVertex();**
окончание примитива: **EndPrimitive();**

Геометрический шейдер

```
#version 430
layout(points) in;
layout(triangle_strip, max_vertices = 40) out;
uniform mat4 MatrWVP;
uniform float Time;
uniform vec3 CamRight;
uniform vec3 CamUp;
out vec2 FDrawTexCoord;
void main( void )
{
  int n = 10;
  float t = Time;
  float s = 3.24;
  vec3 p = gl_in[0].gl_Position.xyz;
  for (int i = 0; i < n; i++)
  {
    float angle = Time * i;
    float si = sin(angle), co = cos(angle);
    vec3 r = CamRight * co + CamUp * si;
    vec3 u = CamRight * -si + CamUp * co;
```

```
    p += vec3(0, s / 2, 0);
    gl_Position = MatrWVP * vec4(p + r * -s + u * s, 1);
    FDrawTexCoord = vec2(0, 1);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * -s + u * -s, 1);
    FDrawTexCoord = vec2(0, 0);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * s + u * s, 1);
    FDrawTexCoord = vec2(1, 1);
    EmitVertex();
    gl_Position = MatrWVP * vec4(p + r * s + u * -s, 1);
    FDrawTexCoord = vec2(1, 0);
    EmitVertex();
    EndPrimitive();
  }
}
```



Геометрический шейдер: пример - частицы

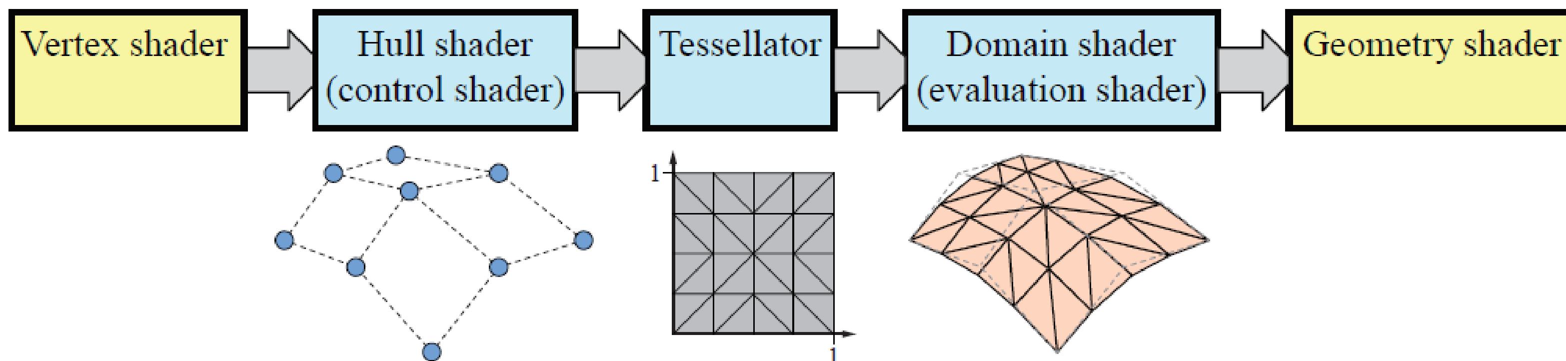
Вершинный шейдер

```
#version 430

layout(location = 0) in vec3 InPosition;

void main( void )
{
  gl_Position = vec4(InPosition, 1);
}
```

Фрагментный шейдер

```
#version 430

layout(location = 0) out vec4 OutColor;

uniform float Time;
uniform bool IsWireframe;
layout(binding = 0) uniform sampler2D Texture0;

in vec2 FDrawTexCoord;

void main( void )
{
  vec4 c = texture2D(Texture0, FDrawTexCoord);

  if (IsWireframe)
    c = vec4(1, 1, 0.5, 1);
  OutColor = c;
}
```

Тесселляционный шейдер




```
Vertex data ->
  Vertex shader ->
    Tessellation control shader ->
      Tessellation evaluation shader ->
        Geometry shader ->
          Rasterizer (assembling primitives, rasterization) ->
            Fragment shader ->
              Raster operation (stencil, alpha, scissor, depth, blending) ->
                Framebuffer
```

```
Input: * (GL_PATCHES)
Output: layout(isolines) out;
        layout(triangles) out;
        layout(quads) out;
```

Тесселляционный шейдер

Тип примитива OpenGL:

```
GL_PATCHES
```

Указываем количество вершин в наборе:

```
glPatchParameteri(GL_PATCH_VERTICES, N);
```

При выводе:

```
. . .
glDrawElements(GL_PATCHES, NumberOfIndices, GL_UNSIGNED_INT, NULL);
  или
glDrawArrays(GL_PATCHES, 0, NumberOfVertices);
. . .
```

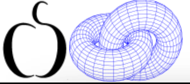
Vertex shader (.VERT):*

```
#version 430

layout(location = 0) in vec3 InPosition;

void main( void )
{
  gl_Position = vec4(InPosition, 1);
}
```

Тесселляционный шейдер: пример - кривая Безье

Tesselation control shader (.CTRL):*

```
#version 430

layout(vertices = 4) out; // Количество вершин в одном наборе

void main( void )
{
  /* Контрольный шейдер вызывается для каждой вершины набора (у нас в нем 4-е вершины
   * смотри layout. Порядковый номер вырщины находится в номере gl_InvocationID
   * gl_in и gl_out - массивы со входными и выходными данными по всем вершинам набора
   * как и в геометрическом шейдере. Мы просто копируем вершину дальше */
  gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
  gl_TessLevelOuter[0] = 1;  // Указываем количество изолиний
  gl_TessLevelOuter[1] = 47; // Указываем величину разбиения ломаной
}
```



Тесселляционный шейдер: пример - кривая Безье

Tesselation evaluation shader:

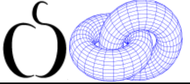
```
#version 430

/* Указываем тип построения разбиения - ломаная (изолиния) */
layout(isolines) in;

uniform mat4 MatrWVP;

vec3 Bezier( vec3 P0, vec3 P1, vec3 P2, vec3 P3, float t )
{
  return P0 * (1 - t) * (1 - t) * (1 - t) +
         P1 * 3 * (1 - t) * (1 - t) * t +
         P2 * 3 * (1 - t) * t * t +
         P3 * t * t * t;
}
-- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
void main( void )
{
  /* Получаем по 4 точки из массива gl_in. Переменная gl_TessCoord в координатах
   * содержит интерполяционный параметр. У нас линия, одно измерение используем только
   * gl_TessCoord.x. В случае бикубических порверхностей - использовались бы
   * gl_TessCoord.x и gl_TessCoord.y. Каждый параметр пробегает от 0 до 1 с шагом,
   * заданным разбиением в контрольном шейдере. Вычислительный шейдер вызывается для
   * каждой точки тесселяции (подразбиения) персонально. */
  gl_Position = MatrWVP *
    vec4(Bezier(gl_in[1].gl_Position.xyz,
                gl_in[1].gl_Position.xyz +
                  (gl_in[2].gl_Position.xyz - gl_in[0].gl_Position.xyz) / 6,
                gl_in[2].gl_Position.xyz +
                  (gl_in[1].gl_Position.xyz - gl_in[3].gl_Position.xyz) / 6,
                gl_in[2].gl_Position.xyz,
                gl_TessCoord.x), 1);
}
```

Тесселляционный шейдер: пример - кривая Безье

Fragment shader:

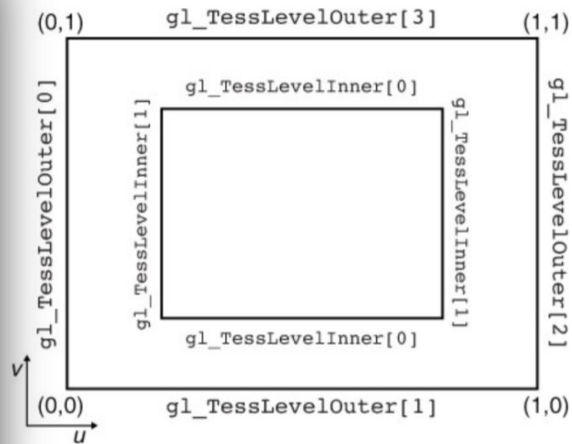
```
#version 430

layout(location = 0) out vec4 OutColor;

void main( void )
{
  OutColor = vec4(1, 0, 1, 1); // Рисуем малиновую точку
}
```



Тесселляционный шейдер: пример - поверхность

Tesselation control shader (.CTRL):*

```
#version 430

layout(vertices = 16) out;

uniform int Addon0;
uniform int Addon1;
uniform int Addon2;

void main( void )
{
  gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;

  // Outer split values
  gl_TessLevelOuter[0] = Addon0;  // by U0
  gl_TessLevelOuter[1] = Addon0;  // by V0
  gl_TessLevelOuter[2] = Addon0;  // by U1
  gl_TessLevelOuter[3] = Addon0;  // by V1

  // Inner split values
  gl_TessLevelInner[0] = Addon1;  // by U
  gl_TessLevelInner[1] = Addon2;  // by V
}
```



Tesselation evaluation shader: Tessellation control shader (.CTRL):*

```
#version 430

layout(quads) in;

uniform mat4 MatrWVP;

void main( void )
{
  float u =  gl_TessCoord.x, v =  gl_TessCoord.y;

  vec3 p = ...
      точка на поверхности
      опорные точки (4x4) - gl_in[i * 4 + j].gl_Position.xyz
      параметры - u, v

  gl_Position = MatrWVP * vec4(p, 1);
}
```

FBO – Frame Buffer Object

Frame Buffer Object (FBO) – Render Target

- создать дескриптор FBO
- создать текстуру для FBO (или несколько текстур) и закрепить за FBO
- создать буфер глубины и закрепить за FBO

FBO – Frame Buffer Object

```
/* FBO depended data */
unsigned
  VG4_RndFBO,          /* Frame buffer object descriptor */
  VG4_RndRenderTex,    /* Render target texture */
  VG4_RndDepthBuf;     /* Render target buffer */
```

FBO – Frame Buffer Object

```
RndTargetsInit:
  /* Fragment shader texture attachment list */
  UINT DrawBuffers[] = {GL_COLOR_ATTACHMENT0};

  /* Create and bind FBO */
  glGenFramebuffers(1, &VG4_RndFBO);
  glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);

  /* Create and bind render texture */
  glGenTextures(1, &VG4_RndRenderTex);
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, VG4_RndRenderTex);
  glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGBA8, VG4_RndFrameW, VG4_RndFrameH);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

  /* Link render texture to FBO */
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, VG4_RndRenderTex, 0);
```

...

FBO – Frame Buffer Object

…

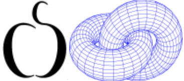
```
/* Create and bind depth buffer */
glGenRenderbuffers(1, &VG4_RndDepthBuf);
glBindRenderbuffer(GL_RENDERBUFFER, VG4_RndDepthBuf);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, VG4_RndFrameW, VG4_RndFrameH);

/* Link depth buffer to FBO */
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, VG4_RndDepthBuf);

/* Set fragment shader texture attachment list:
 * array contents layout slot redirection for every output value */
glDrawBuffers(1, DrawBuffers);

if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
  return;

/* Go back to default render buffer */
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

FBO – Frame Buffer Object

```
RndTargetsClose:
  /* Delete FBO primitive */
  VG4_RndPrimFree(&VG4_RndFBOFramePrim);
  /* Bind FBO */
  glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);
  /* Unlink render texture from FBO */
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
  /* Delete render texture */
  glDeleteTextures(1, &VG4_RndRenderTex);
  /* Unlink depth buffer from FBO */
  glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
  /* Delete depth buffer */
  glDeleteRenderbuffers(1, &VG4_RndDepthBuf);
  /* Unlink FBO */
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
  /* Delete FBO */
  glDeleteFramebuffers(1, &VG4_RndFBO);
```

FBO – Frame Buffer Object

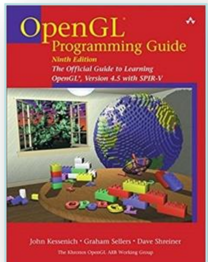
```
RndTargetsStart:  // вызываем из RndStart()
  /* Bind FBO */
  glBindFramebuffer(GL_FRAMEBUFFER, VG4_RndFBO);

  /* Set viewport */
  glViewport(0, 0, VG4_RndFrameW, VG4_RndFrameH);
  glClearColor(0.3, 0.5, 0.7, 1);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```



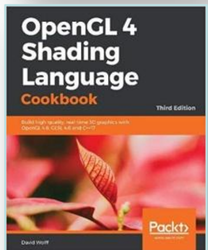
FBO – Frame Buffer Object


```
RndTargetsEnd:  // вызываем из RndEnd()
  /* Set default FBO */
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
  glViewport(0, 0, VG4_RndFrameW, VG4_RndFrameH);

  /* Draw post frame primitive (full screen rectangle) to screen */
  glUseProgram(шейдер обработки FBO);

  /* Send render target */
  glActiveTexture(GL_TEXTURE3); // в примере используется 3
  glBindTexture(GL_TEXTURE_2D, VG4_RndRenderTex);

  /* Draw frame */
  glPushAttrib(GL_ALL_ATTRIB_BITS); // запоминаем состояние атрибутов
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_BLEND);
  glDisable(GL_CULL_FACE);
  ... выводим полноэкранный примитив FBO ...
  glPopAttrib(); // восстанавливаем состояние атрибутов
  glUseProgram(0);
```

Литература



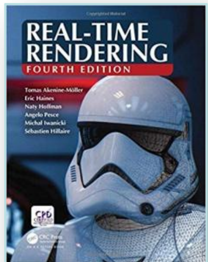
John Kessenich, Graham Sellers, Dave Shreiner,
«OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition)»,
Addison-Wesley Professional, 2016.



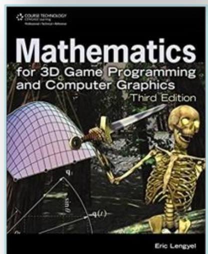
David Wolff, ***«OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++17, 3rd Edition»***,
Packt Publishing, 2018.

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

Литература



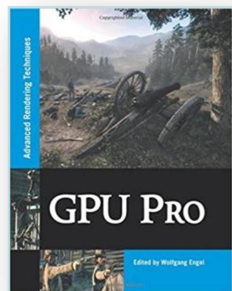
Tomas Akenine-Möller, Eric Haines, Naty Hoffman,
«Real-Time Rendering, Fourth Edition»,
A K Peters/CRC Press, 2018.

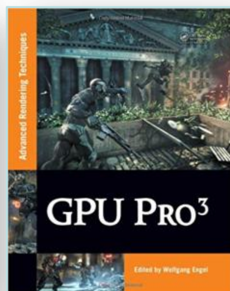


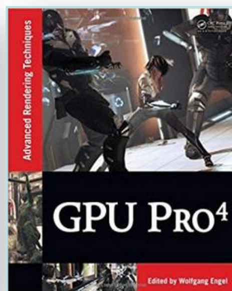
Eric Lengyel,
«Mathematics for 3D Game Programming and Computer Graphics, Third Edition 3rd Edition»,
Cengage Learning PTR, 2011

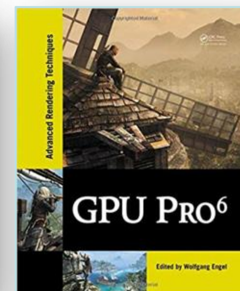
Elmar Eisemann, Michael Schwarz, Ulf Assarsson, Michael Wimmer,
«Real-Time Shadows»,
A K Peters/CRC Press, 2011.

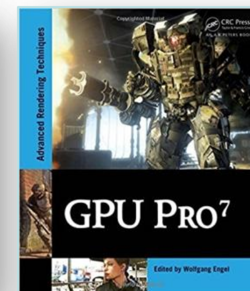
Литература











Wolfgang Engel (Editor), ***«GPU Pro: Advanced Rendering Techniques»***, A K Peters/CRC Press, 2010.

«GPU Pro2», 2011; ***«GPU Pro3»***, 2012; ***«GPU Pro4»***, 2013; ***«GPU Pro5»***, 2014; ***«GPU Pro6»***, 2015; ***«GPU Pro7»***, 2016.

Wolfgang Engel, ***«GPU Zen: Advanced Rendering Techniques»***, Bowker Identifier Services, 2017