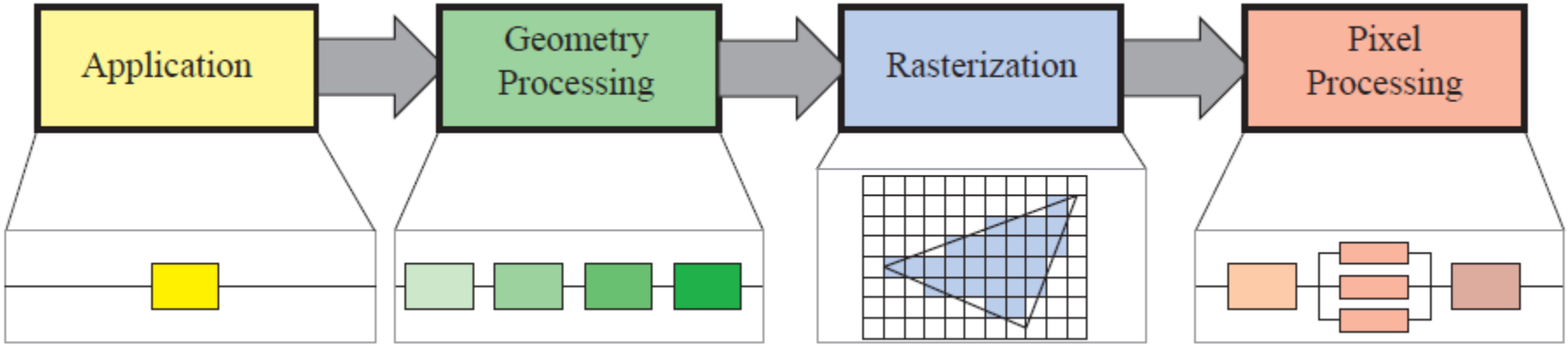
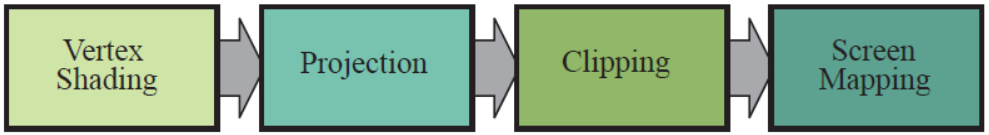
Графика реального времени. OpenGL. Часть 2

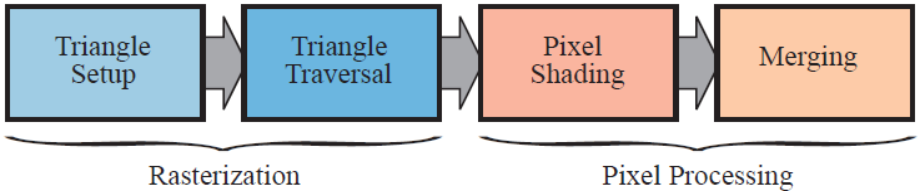
материалы занятий: https://compsciclub.ru/courses/graphics2018/2018-autumn/classes/
дублируются на сайте: http://www.school30.spb.ru/cgsg/cgc2018/

Графический конвейер

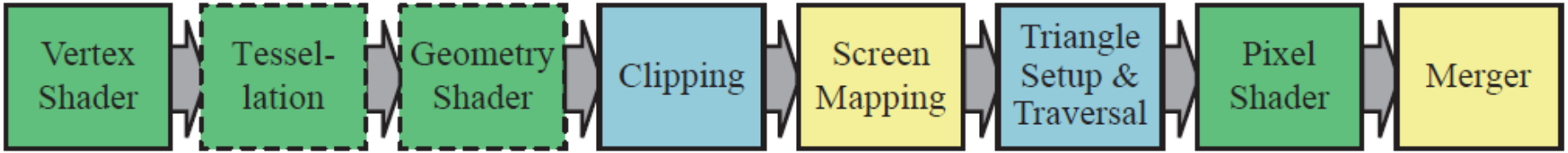




Geometry Processing



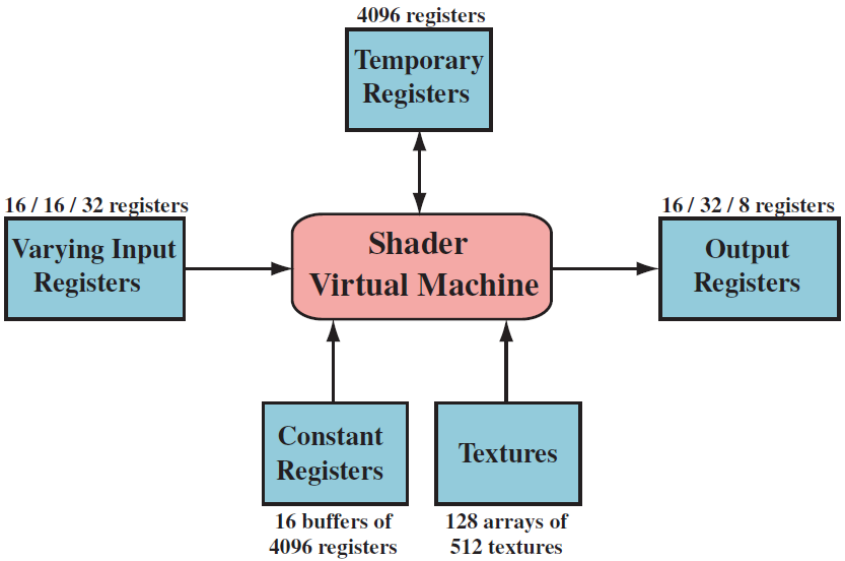
Графический конвейер




```
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
```

Шейдера

Cg (NVidia, 2004)
GLSL (OpenGL)
HLSL (Direct3D)



`CPU -> Vertex attributes -> VS`

-- передаем для каждой вершины набор атрибутов (позиция, цвет и т.п.) через VBO

`shader -> Varying variables -> shader`

-- переменные от каждой вершины могут быть переданы для дальнейшей обработки, например, после вершинного шейдера они интерполируются от вершины к вершине во время растеризации и строятся для каждого пикселя и приходят во фрагментный шейдер, от фрагментного шейдера выходные данные поступают в буфер кадра

`Uniform variables – globals`

-- глобальные переменные, доступные во всех шейдерах одной шейдерной программы (например, матрица преобразований, время, текстурные сэмплеры, параметры освещения и т.п.)

Основы GLSL

Базовые типы:

- `bool` - `true` | `false`
- `int`, `uint`
- `sampler` (`sampler1D`, `sampler 2D`, `sampler3D`)

- `float`

структурные:

- Vectors:
  - `bvec2`, `bvec3`, `bvec4` (bool)
  - `ivec2`, `ivec3`, `ivec4` (int)
  - `uvec2`, `uvec3`, `uvec4` (uint)
  - `vec2`, `vec3`, `vec4` (float)
  - `dvec2`, `dvec3`, `dvec4` (double)
- Matrices:
  - `mat2`, `mat3`, `mat4` (2x2, 3x3, 4x4)
  - `mat2x3`, `mat2x4`, `mat3x2`, `mat3x4`, `mat4x2`, `mat4x3` (rows x columns)
  - `dmat2`, `dmat3`, `dmat4`
  - `dmat2x3`, `dmat2x4`, `dmat3x2`, `dmat3x4`, `dmat4x2`, `dmat4x3`

Основы GLSL

Запись числовых констант:

1.5 – float
1.5f – float
1.5lf – double

Инициализация:

```
float a = 13.47;
bool is_space = false;
ivec3 a = ivec3(1, 2, 3);
~~uvec3 b = uvec3(1, -2, 3);~~
vec3 v = vec3(1.0, 2.0, 3);
vec4 v1 = vec4(1.0, vec3(4, 5, 6));
vec4 v2 = vec4(vec3(4, 5, 6), 8);
vec4 v3 = vec4(vec2(4, 5), vec2(13, 8));
vec4 v3 = vec4(0);    <=>    vec4(0, 0, 0, 0);
```

заполнение матрицы по **столбцам**:

```
mat3 m = mat3(vec3(1, 2, 3),
              vec3(3, 4, 5),
              vec3(6, 7, 8));
```

Основы GLSL

Доступ к компонентам векторов:

```
vec2 p;
p[0] = 1.0;   <-- x
p.x = 1;
p.y = 3.3;
```

любой вектор – это набор полей:

```
{x, y, z, w} - геометрия
{r, g, b, a} - цвет
{s, t, p, q} – текстурирование
p.x == p.r == p.s
```

swizzle:

```
vec3 v = p.xxy;  <=> vec3(p.x, p.x, p.y);
vec4 c = v.rrrr;
~~vec4 r = v.rryy;~~
```

Доступ к компонентам матриц:

```
mat4 m;
m[0] = vec4(1); -- весь первый столбец в 1
m[2][1] = 30.50;
```

```
vec4 v3 = vec4(vec2(4, 5), vec2(13, 8));
```

Операции:

```
mat3 T, R, M;
vec3 v, b;
float f;
b = v + f;
  (b.x = v.x + f, b.y = v.y + f, ... )
```

!!! используется нотация вектор-столбец:

```
b = M * v;
M = R * T;
b = R * T * v;
```

преобразования скалярных типов:

```
int a = int(47.13);
bool c = false;
float x = float(c);  <-- 0.0
```

Основы GLSL

Агрегатные типы:

структуры:

```
struct Light
{
  vec3 Pos;
  vec4 Color;
  float Attenuation;
};
Light L = Light(vec3(8, 8, 8), vec4(1, 0, 0, 1), 0.47);
```

массивы:

```
Light ls[10];
for (int i = 0; i < 10; i++)
{
  ls[i].Attenuation = i / 10.0;
}
for (int i = 0; i < ls.length(); i++)
  . . .
vec4 v;
v.length() --> 4
```

Препроцессор:

```
#error
#pragma
#version
#define
#if #ifdef #ifndef #else #endif #elif
```

Точка входа:

```
void main( void )
{
}
```

Основы GLSL

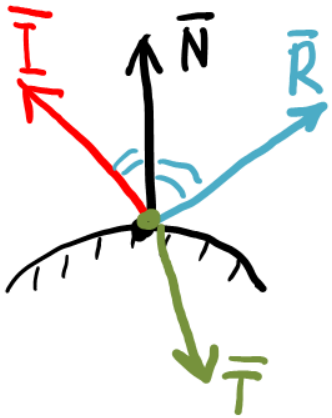
Дополнительные функции:

```
float t = dot(V1, V2);
vec3 V3 = cross(V1, V2);

T X = max(X1, X2)
T X = min(X1, X2)

T X = clamp(X1, A, B) <=> min(max(X1, A), B) <=> (X1 < A ? A : X1 > B : B : X)
T X = mix(A, B, t)    <=> A * (1 - t) + B * t

sin cos tan asin acos atan(x) atan(y, x) pow(x, y) log
exp sqrt inversesqrt abs sign floor round trunc ceil mod

float t = length(V1)
float t = distance(P1, P2)
vec V = normalize(V1)

mat M;
mat T = inverse(M)
mat T = transpose(M)
float d = determinant(M)
```

Основы GLSL

Функции для моделирования освещения:

```
vec V = faceforward(N, I, Nref) <=>  dot(I, Nref) < 0 ? N : -N
vec R = reflect(I, N)            <=>  I - 2 * dot(N, I) * N
vec T = refract(I, N, eta)       <=>  k = 1.0 - eta * eta * (1.0 - dot(N, I) * dot(N, I));
                                      if (k < 0.0)
                                        V = vec(0.0);
                                      else
                                        V = eta * I - (eta * dot(N, I) + sqrt(k)) * N;
```



Справочная таблица: https://www.khronos.org/files/opengl46-quick-reference-card.pdf

Загрузка шейдера

```
int res, NumOfShaders,
    shd_type[] =
    {
      GL_VERTEX_SHADER, GL_TESS_CONTROL_SHADER,
      GL_TESS_EVALUATION_SHADER, GL_GEOMETRY_SHADER,
      GL_FRAGMENT_SHADER
    };
unsigned prg = 0, shd[5];
char *txt, buf[1000];

/* Load text file */
txt = LoadTextFile("a.vert");
/* Create shader */
shd[0] = glCreateShader(shd_type[0]);
/* Attach text to shader  */
glShaderSource(shd[0], 1, &txt, NULL);
free(txt);
/* Compile shader */
glCompileShader(shd[i]);
glGetShaderiv(shd[i], GL_COMPILE_STATUS, &res);
if (res != 1)
{
  glGetShaderInfoLog(shd[0], sizeof(buf), &res, buf);
  ErrorLog(buf);
  return;
}
. . .
```

```
/* Create program */
prg = glCreateProgram();
/* Attach shaders to program */
for (i = 0; i < NumOfShaders; i++)
  glAttachShader(prg, shd[i]);
/* Link program */
glLinkProgram(prg);
glGetProgramiv(prg, GL_LINK_STATUS, &res);
if (res != 1)
{
  glGetProgramInfoLog(prg, sizeof(buf), &res, buf);
  ErrorLog(buf);
  return;
}
return prg;
```

Удаление шейдера:

```
  unsigned i, n, shds[5];

  glGetAttachedShaders(Prg, 5, &n, shds);
  for (i = 0; i < n; i++)
  {
    glDetachShader(Prg, shds[i]);
    glDeleteShader(shds[i]);
  }
  glDeleteProgram(Prg);
```

Простейший шейдер

vert.glsl

```
// версия языка шейдера (4.5)
#version 450

// кто куда приходит
layout(location = 0) in vec3 InPosition;
layout(location = 1) in vec2 InTexCoord;
layout(location = 2) in vec3 InNormal;
layout(location = 3) in vec4 InColor;

// глобальные переменные (произведение
// матриц: World * View * Proj)
uniform mat4 MatrWVP;

// выходные параметры (varying)
out vec4 DrawColor;

void main( void )
{
  gl_Position = MatrWVP * vec4(InPosition, 1);
  DrawColor = InColor;
}
```

frag.glsl

```
// версия языка шейдера (4.5)
#version 450

// выходные параметры - цвет рисования
layout(location = 0) out vec4 OutColor;

// входные параметры (varying)
in vec4 DrawColor;

void main( void )
{
  OutColor = DrawColor;
}
```

Uniforms

```
int loc;

glUseProgram(PrgId);

if ((loc = glGetUniformLocation(PrgId, "Matrix")) != -1)
  glUniformMatrix4fv(loc, 1, FALSE, указатель на первый элемент матрицы);

if ((loc = glGetUniformLocation(PrgId, "Vector")) != -1)
  glUniform3fv(loc, 1, указатель на первую координату вектора);

if ((loc = glGetUniformLocation(PrgId, "FloatNumber")) != -1)
  glUniform1f(loc, вещественное число);

if ((loc = glGetUniformLocation(ProgId, "IntegerNumber")) != -1)
  glUniform1i(loc, целое число);

. . . отрисовка . . .

glUseProgram(0);
```

Текстурирование

Генерация «свободных» номеров текстур в массив:

```
int TexNames[4];
glGenTextures(4, TexNames);
```

Переключение текстур (установка номера активной текстуры):

```
glBindTexture(GL_TEXTURE_2D, number);
```

Функции задания изображений в текстуру:

```
glTexImage2D(GL_TEXTURE_2D,
  0,                  уровень MIP (multum in parvo) 0 - базовая картинка
  1,                  количество компонент на точку (1, 2, 3, 4)
  w, h,               ширина и высота (должны быть степенью 2)
  0,                  наличие границы
  GL_LUMINANCE,       трактовка компонент точки (GL_LUMINANCE, GL_BGR_EXT, GL_BGRA_EXT)
  GL_UNSIGNED_BYTE,   тип каждой компоненты (GL_UNSIGNED_BYTE, GL_FLOAT)
  ptr);               указатель на массив с компонентами
```

Автоматическая генерация текстур всех размеров «вниз», начиная с указанного размера

```
gluBuild2DMipmaps(GL_TEXTURE_2D,
  3,                  количество компонент на точку (1, 2, 3, 4)
  w, h,               ширина и высота - любые
  GL_BGR_EXT,         трактовка компонент точки (GL_LUMINANCE, GL_BGR_EXT, GL_BGRA_EXT)
  GL_UNSIGNED_BYTE,   тип каждой компоненты (GL_UNSIGNED_BYTE, GL_FLOAT)
  ptr);               указатель на массив с компонентами
```

Текстурирование

Управление фильтрацией:

```
glTexParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER, GL_NEAREST);
                           GL_LINEAR
                           GL_NEAREST_MIPMAP_NEAREST
                           GL_LINEAR_MIPMAP_NEAREST
                           GL_NEAREST_MIPMAP_LINEAR
                           GL_LINEAR_MIPMAP_LINEAR

glTexParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_MAG_FILTER, GL_NEAREST);
                           GL_LINEAR
```

Параметры наложения (повтор текстуры по направлениям):

```
glTexParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_WRAP_S, GL_REPEAT);
    GL_TEXTURE_WRAP_T  GL_CLAMP
```

Параметры выравнивания - на какое число должна делится нацело длина одной строки в байтах:

для установки изображения в текстуру (из программы в OpenGL)

```
glPixelStorei(GL_UNPACK_ALIGNMENT, число(1,2,4,...));
```

для чтения изображения (от OpenGL в программу)

```
glPixelStorei(GL_PACK_ALIGNMENT, число(1,2,4,...));
```

Текстурирование

Функции работы с новыми форматами:

```
glTexStorage2D(GL_TEXTURE_2D,
               MipLevels,      - количество уровней MipMap
               GL_RGB32F,      - формат RGB float (GL_RGBA32UI, GL_R32F, ...)
               W, H);          - размер текстуры (w, h)
glTexSubImage2D(GL_TEXTURE_2D,
                0,             - уровень MipMap
                0, 0,          - смещение изображения в текстуре (dx, dy)
                W, H,          - размер вставляемого изображения в текстуру (w, h)
                GL_RGB,        - формат RGB – три компоненты
                GL_FLOAT,      - тип компоненты
                ptr);          - сам массив
```

Текстурирование

*Выборка цвета из текстуры осуществляется сэмплерами (**samplers**)*

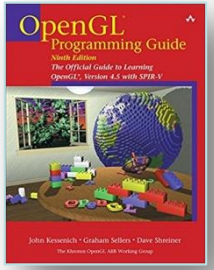
```
glActiveTexture(GL_TEXTURE0 + SamplerNo);  -- активация сэмплера
glBindTexture(GL_TEXTURE_2D, TexId);       -- подключение в него текстуры
```

На шейдере (фрагментном):

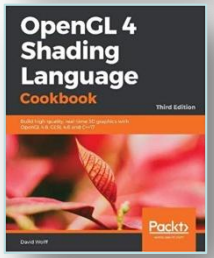
```
// текстурные координаты от вершинного шейдера
in vec2 DrawTexCoord;

// указания какой сэмплер как называется
layout(binding = 0) uniform sampler2D TextureKd;
layout(binding = 1) uniform sampler2D TextureMask;

void main( void )
{
  . . .
  // получение цвета
  vec4 tex_color = texture(TextureKd, DrawTexCoord);
  . . .
}
```

texture ? texelFetch

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.