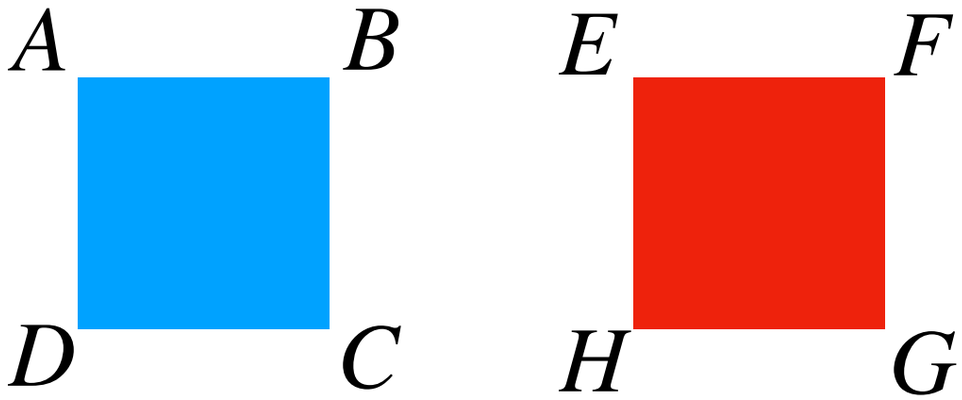
Графический конвейер в OpenGL

Построение трехмерного прообраза

Многоугольники
$ABCD$
$EFGH$

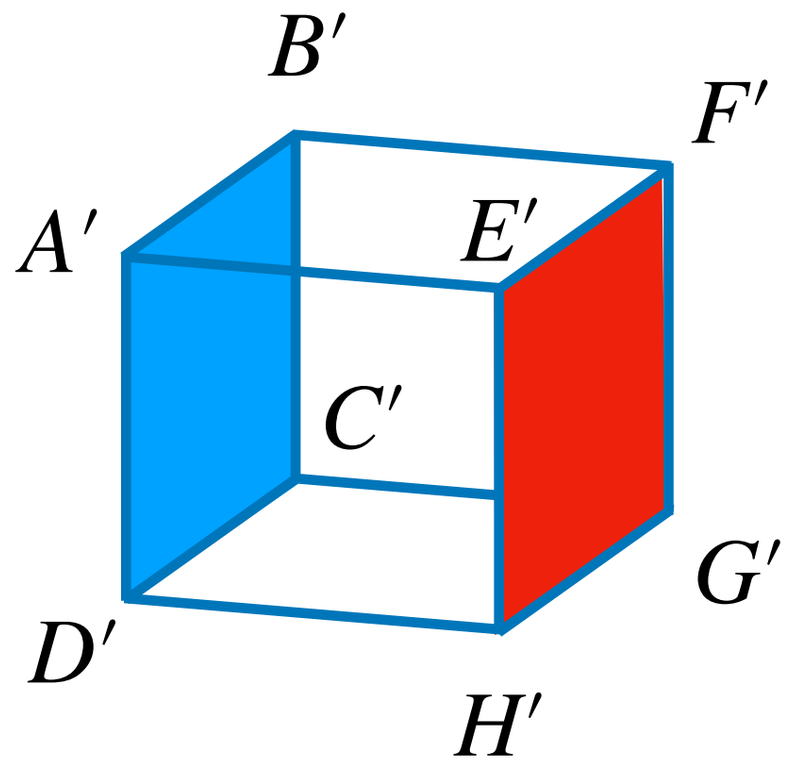


$modelView$ – Модельное преобразование (переход в мировую систему координат)

Построение трехмерного прообраза

Многоугольники
$ABCD$
$EFGH$

Отрезки
AE
BF
CG
DH



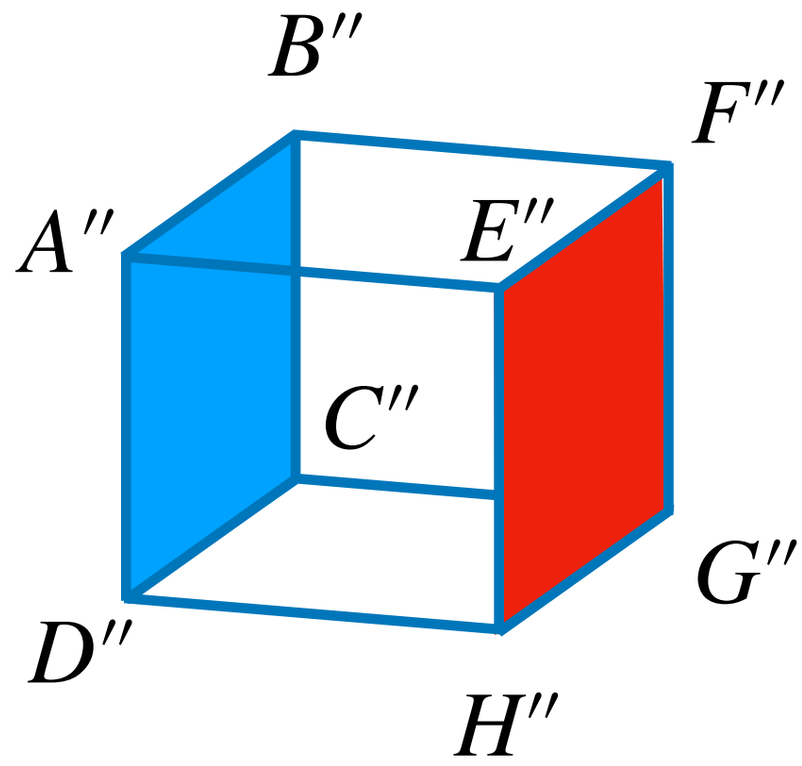
$$A' = modelView \cdot A$$

$modelView$ – Модельное преобразование (переход в мировую систему координат)
$cameraView$ – Переход в систему координат наблюдателя

Прообраз в системе координат наблюдателя

Многоугольники
$ABCD$
$EFGH$

Отрезки
AE
BF
CG
DH



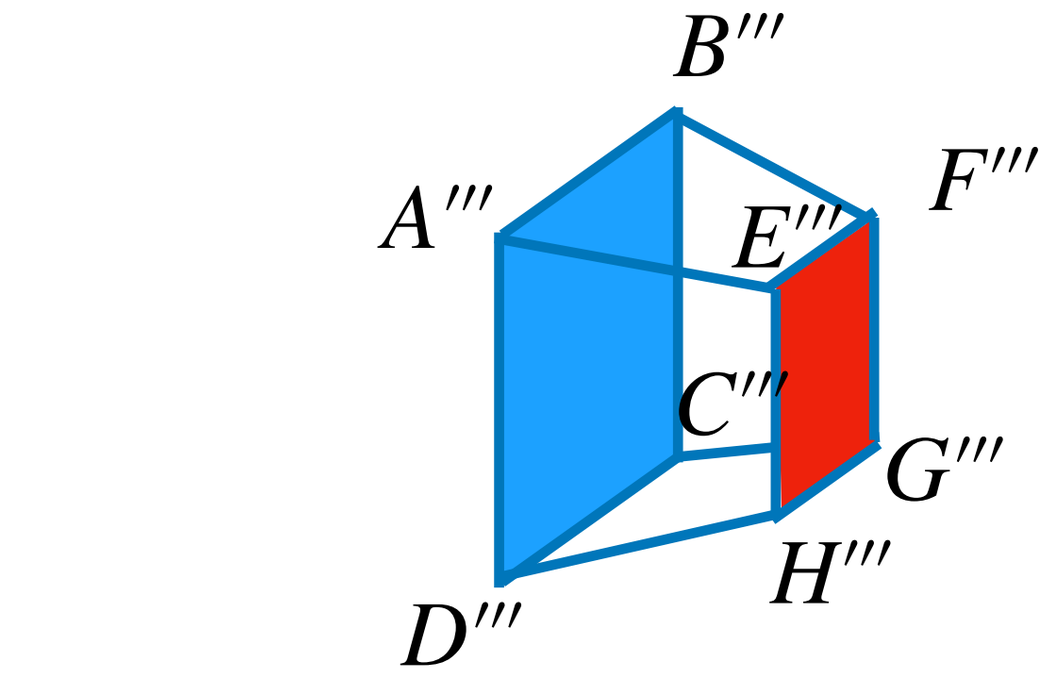
$$A'' = cameraView \cdot A'$$

$modelView$ – Модельное преобразование (переход в мировую систему координат)
$cameraView$ – Переход в систему координат наблюдателя
$clipView$ – Переход в пространство отсечения

Пространство отсечения

Многоугольники
$ABCD$
$EFGH$

Отрезки
AE
BF
CG
DH



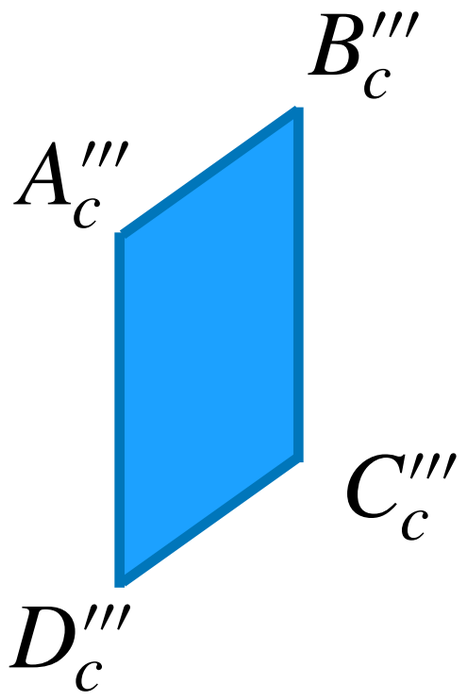
$$A''' = clipView \cdot A''$$

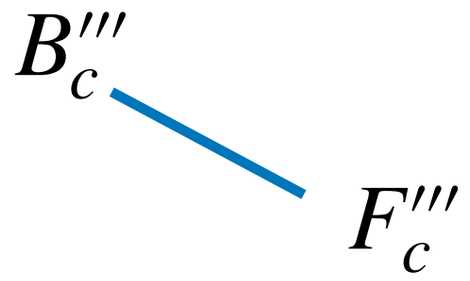
$modelView$ – Модельное преобразование (переход в мировую систему координат)
$cameraView$ – Переход в систему координат наблюдателя
$clipView$ – Переход в пространство отсечения
C – Переход к размерностям целевого изображения (масштабирование и сдвиг)

Растеризация

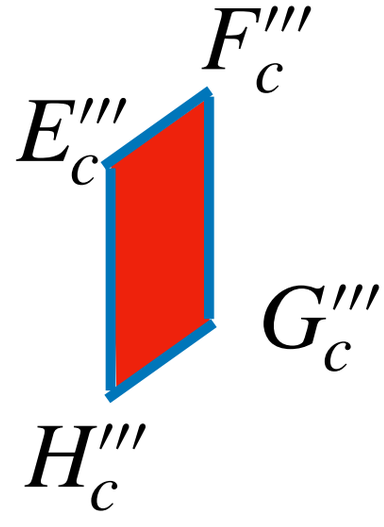
Многоугольники
$ABCD$
$EFGH$

Отрезки
AE
BF
CG
DH





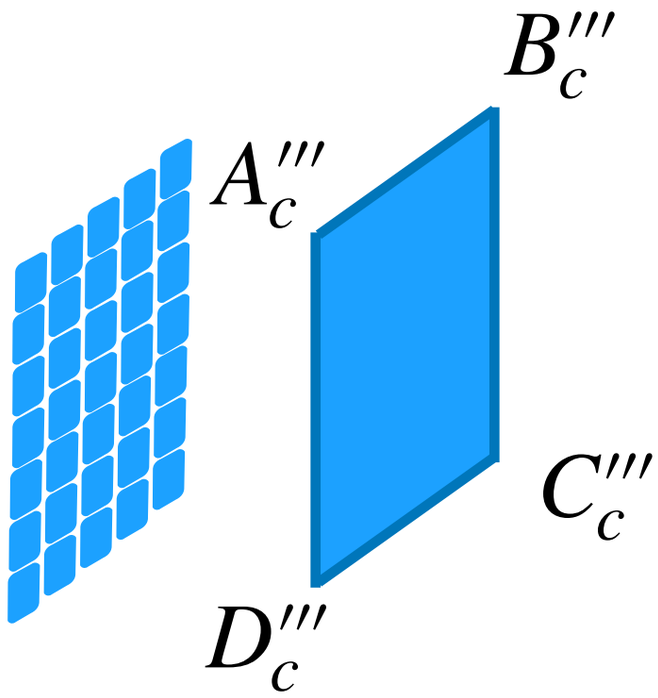
$$A_c''' = C \cdot A'''$$

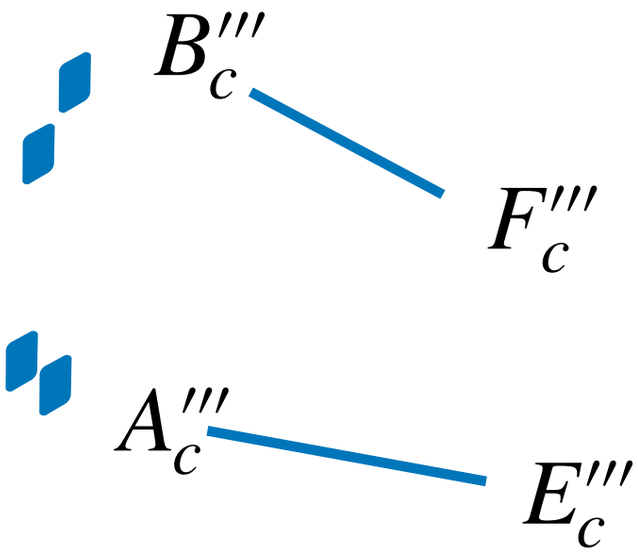


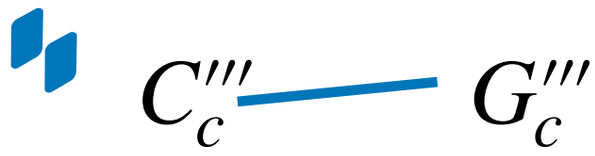
Растеризация

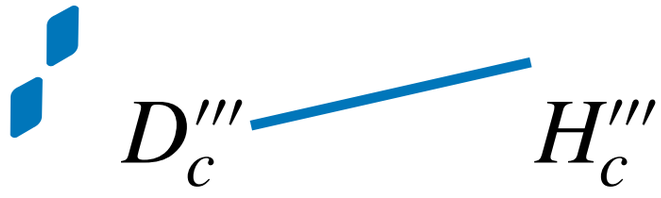
Многоугольники
$ABCD$
$EFGH$

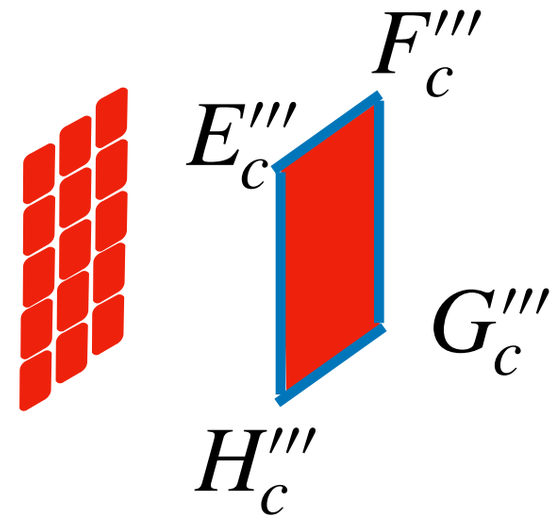
Отрезки
AE
BF
CG
DH



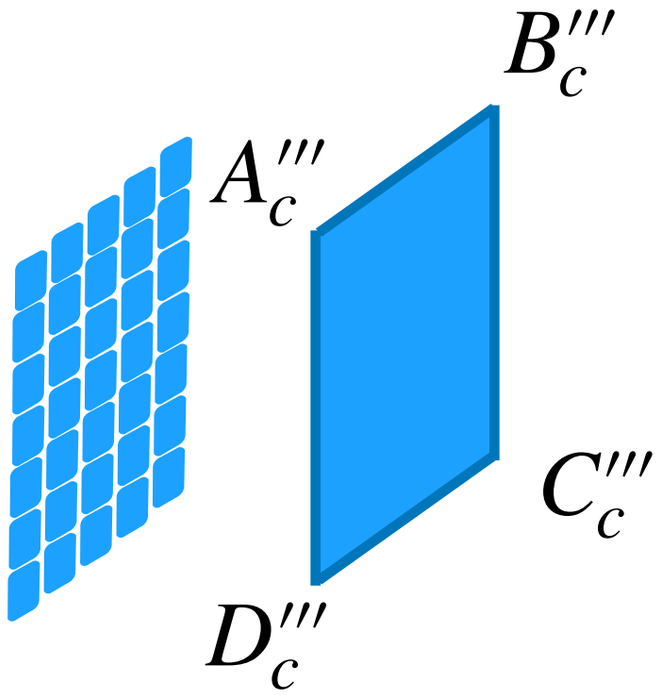


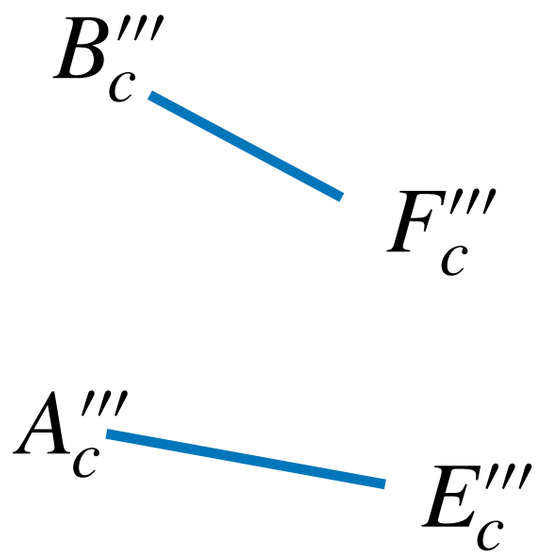






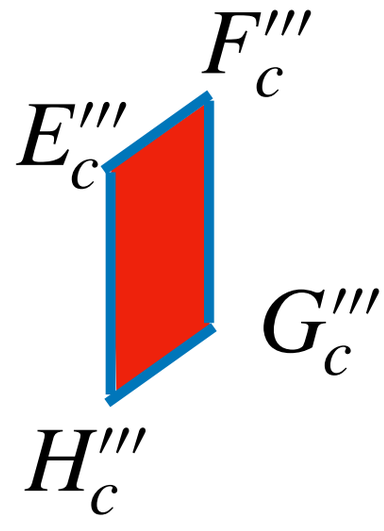
Алгоритм с использованием Z-буфера



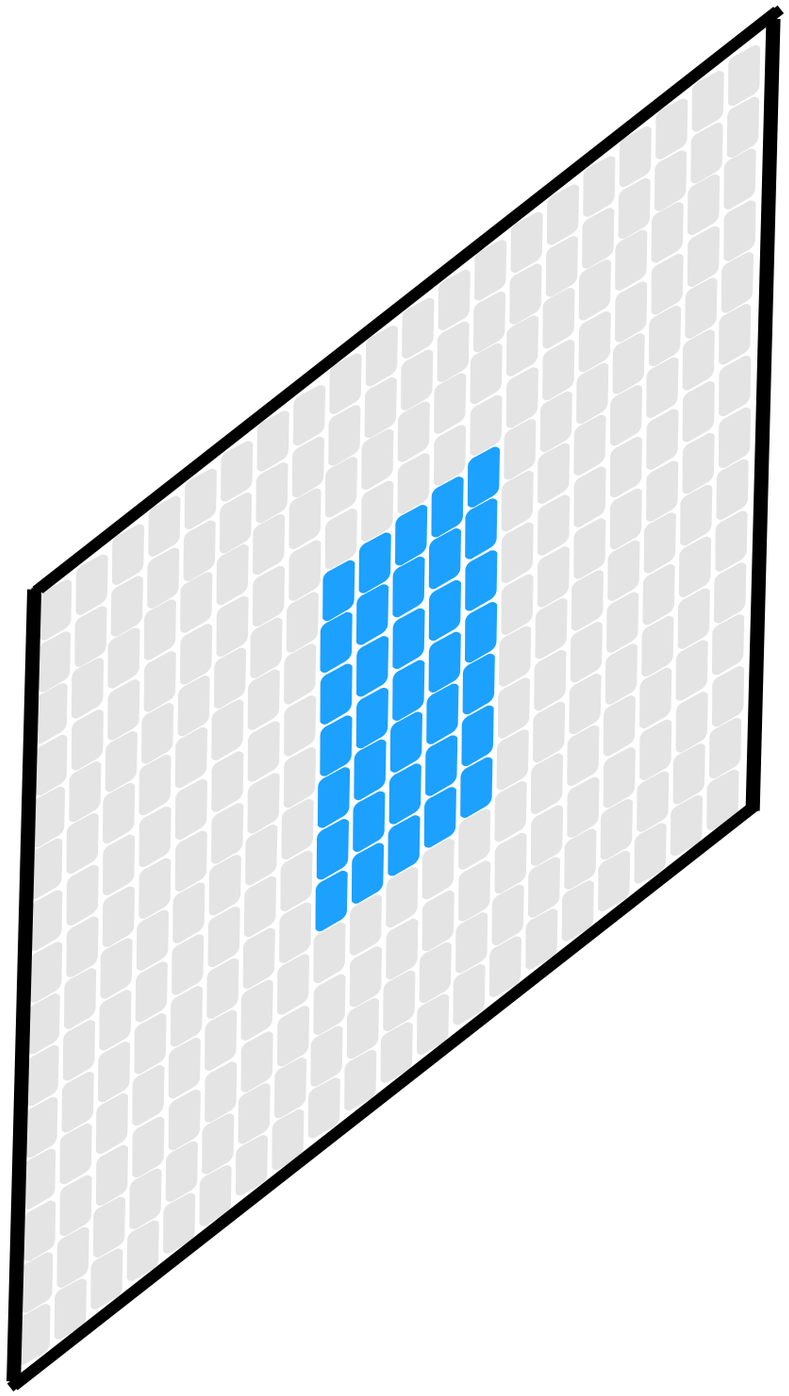


Многоугольники
$ABCD$
$EFGH$

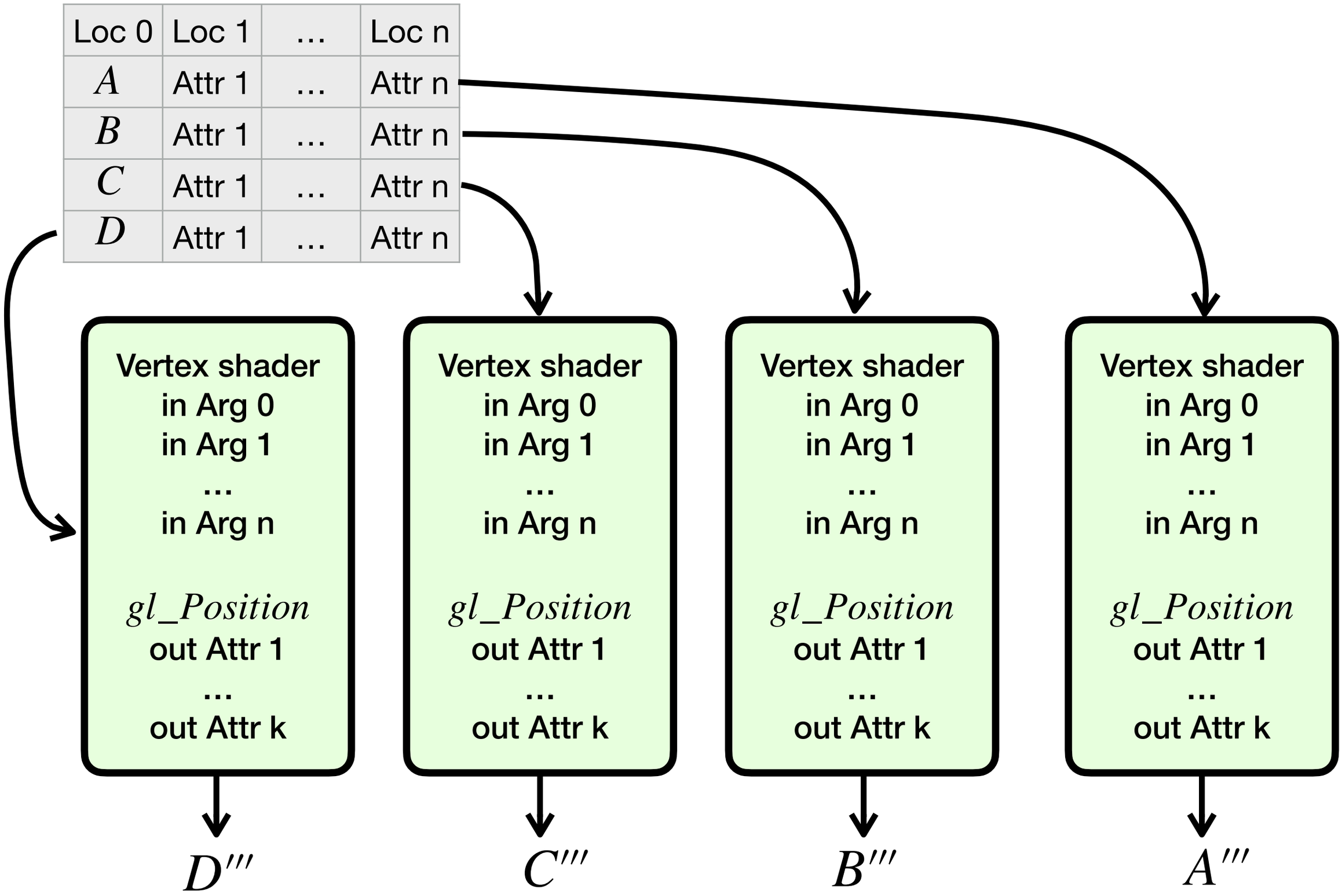
Отрезки
AE
BF
CG
DH



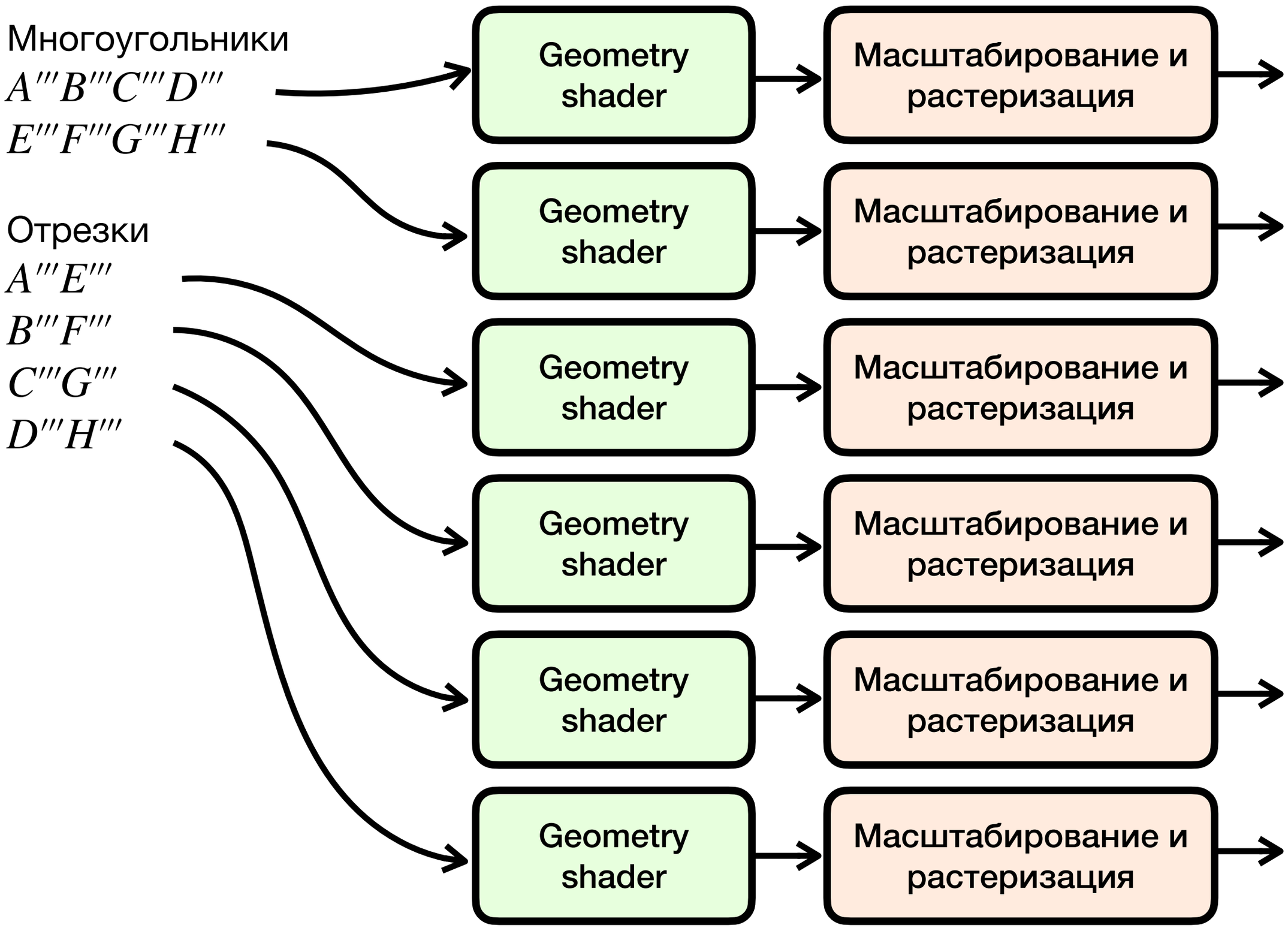
Алгоритм использующий Z-буфер



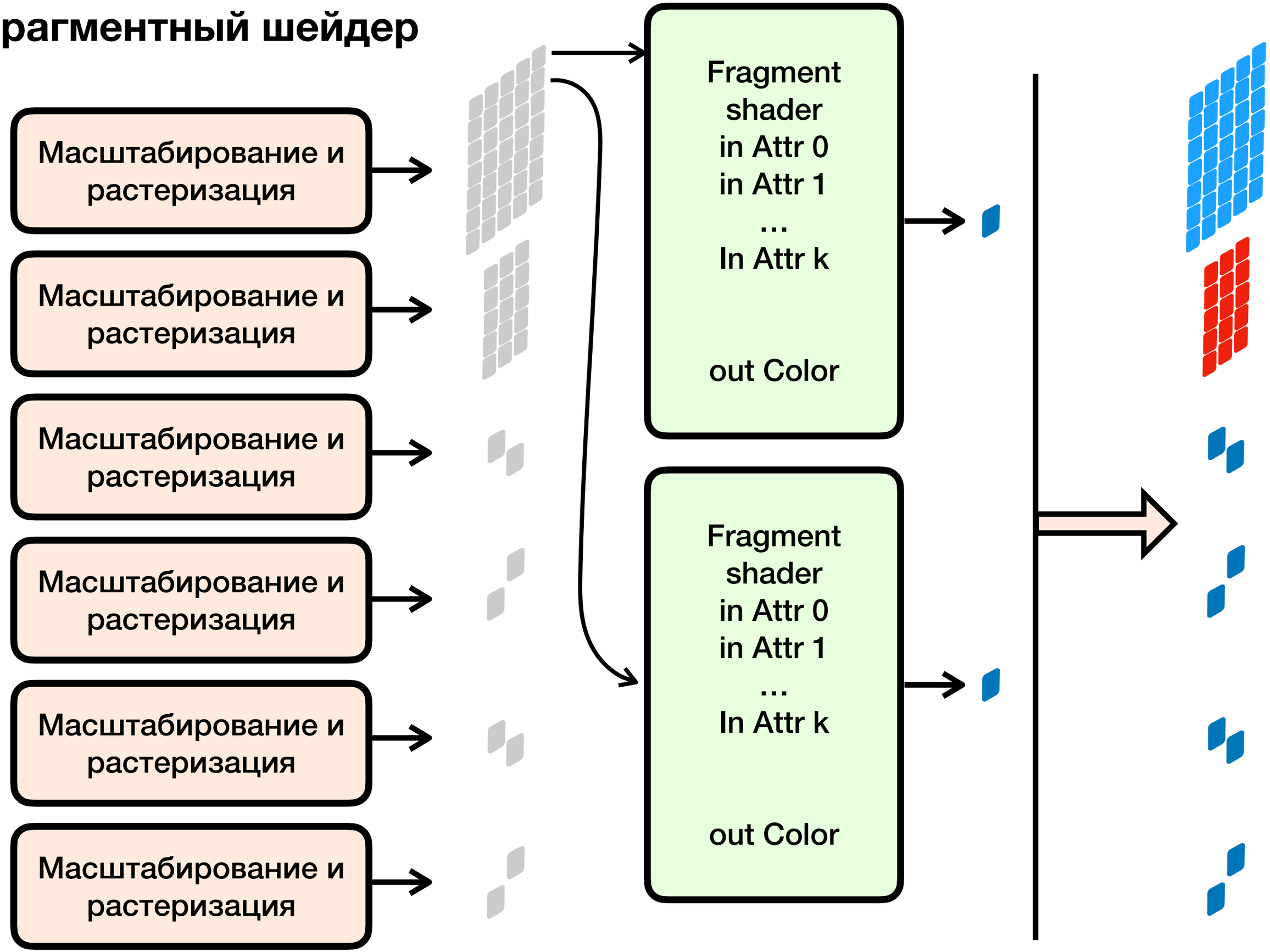
Вершинный шейдер



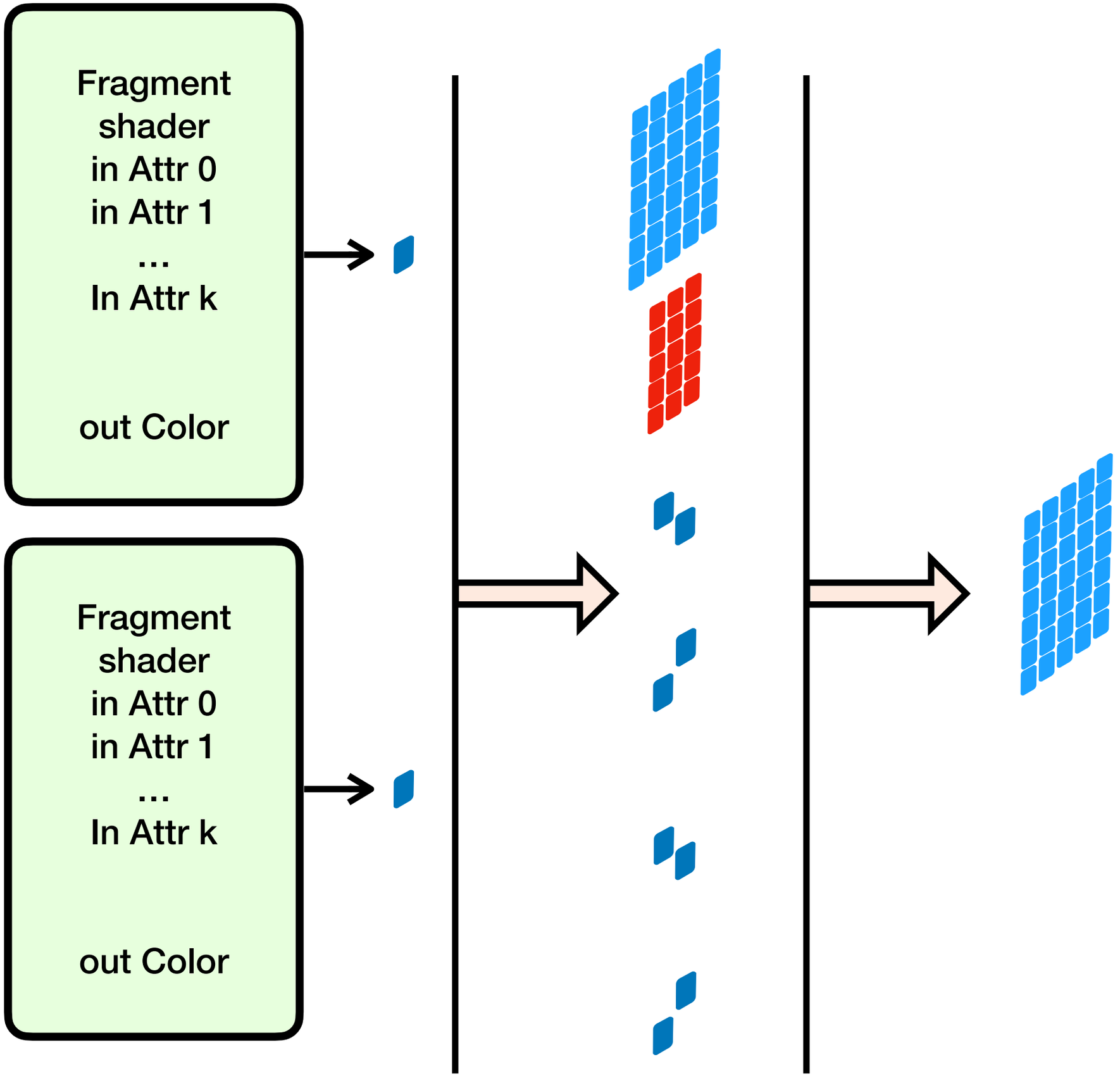
Шейдер геометрии и растеризация



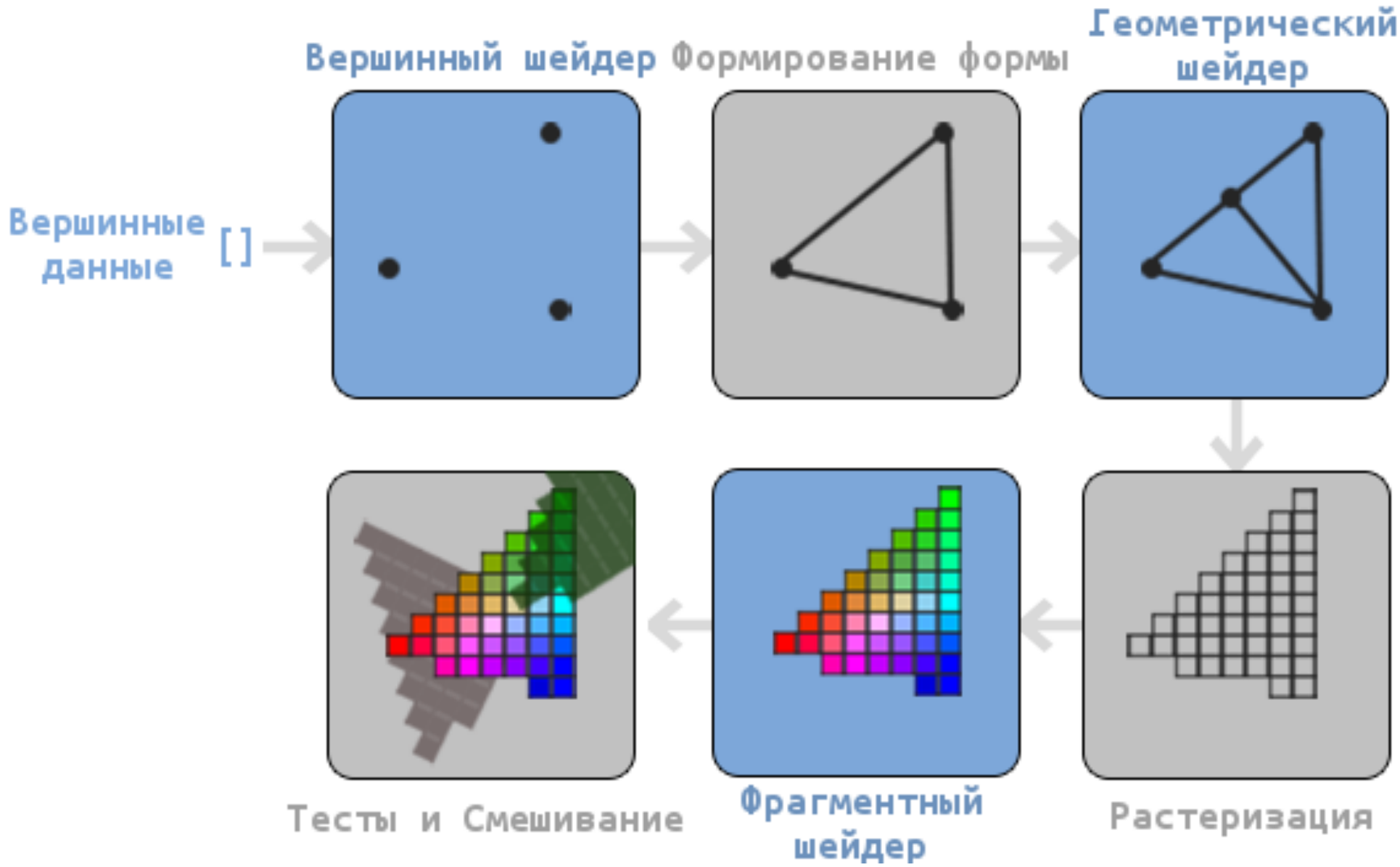
Фрагментный шейдер



Тест глубины

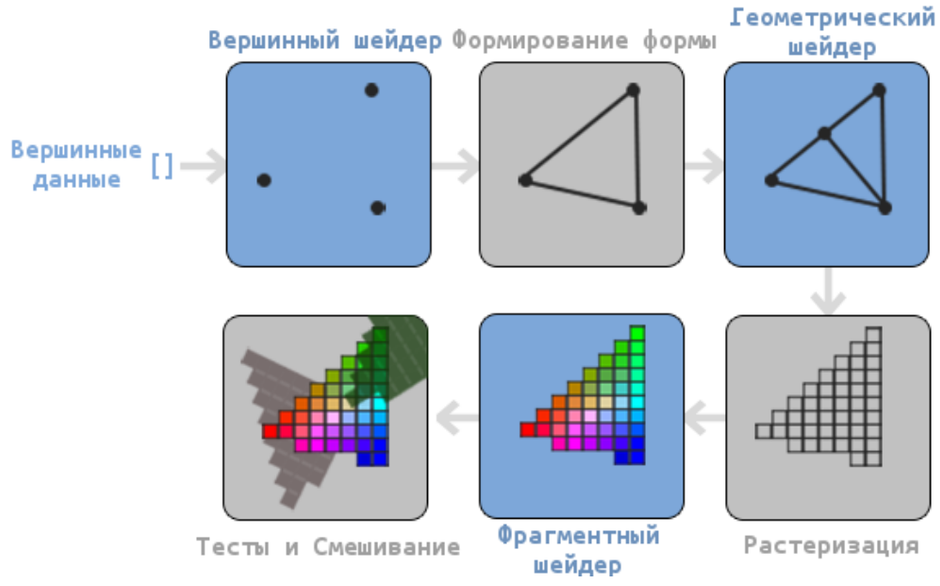


Графический конвейер



Источник диаграммы https://habr.com/ru/post/311808/ – перевод книги Joey de Vries “Learn OpenGL”, оригинал можно найти https://learnopengl.com/

GPU

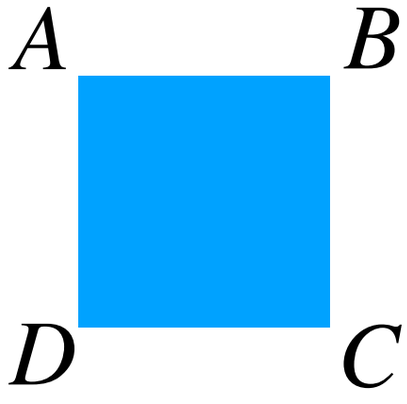


CPU

Подготовка вершинных данных
Пересылка данных на GPU
Подготовка текстов шейдеров
Пересылка текстов шейдеров на GPU
Управление созданием шейдерной программы
Управление отрисовкой

Вершинный массив

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n

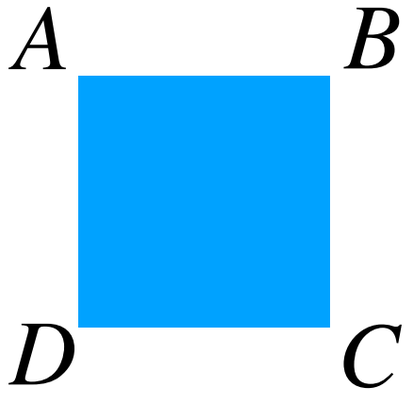


Вершинный буфер GL_ARRAY_BUFFER

A	Attr 1	…	Attr n	B	Attr 1	…	Attr n	C	Attr 1	…	Attr n	D	Attr 1	…	Attr n

Вершинный массив

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n



Вершинный буфер GL_ARRAY_BUFFER

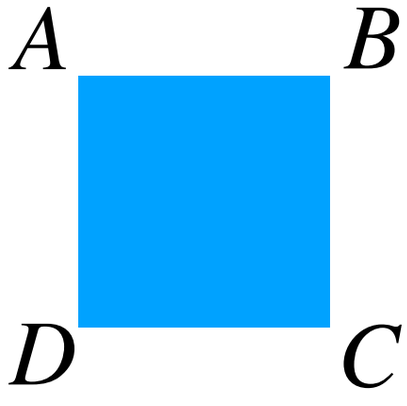
A	Attr 1	…	Attr n	B	Attr 1	…	Attr n	C	Attr 1	…	Attr n	D	Attr 1	…	Attr n
x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3

S

glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, S, (GLvoid*)0)

Вершинный массив

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n



Вершинный буфер GL_ARRAY_BUFFER

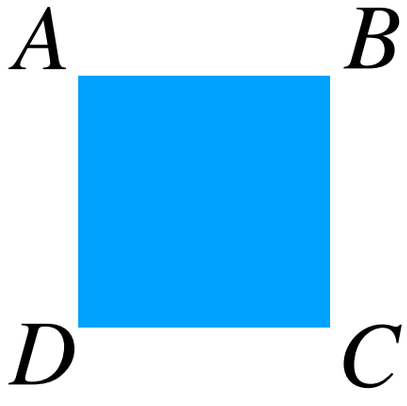
A	Attr 1	…	Attr n	B	Attr 1	…	Attr n	C	Attr 1	…	Attr n	D	Attr 1	…	Attr n
x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3

R_1 S

glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, S, (GLvoid*)R_1)

Вершинный массив

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n



Вершинный буфер GL_ARRAY_BUFFER

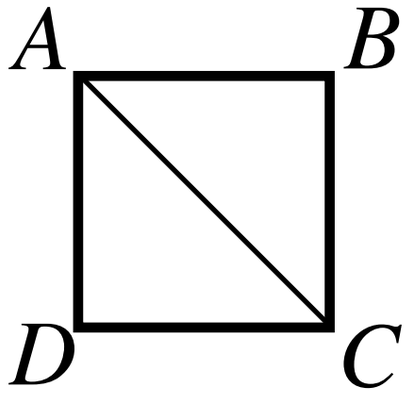
A	Attr 1	…	Attr n	B	Attr 1	…	Attr n	C	Attr 1	…	Attr n	D	Attr 1	…	Attr n
x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3	x,y,z,a	r,g,b		n1,n2,n3

R_n S

glVertexAttribPointer(n, 3, GL_FLOAT, GL_FALSE, S, (GLvoid*)R_n)

Вершинный массив

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n



Вершинный буфер GL_ARRAY_BUFFER

A	Attr 1	…	Attr n	B	Attr 1	…	Attr n	C	Attr 1	…	Attr n	D	Attr 1	…	Attr n

Буфер индексов GL_ELEMENT_ARRAY_BUFFER

0	1	2	0	2	3

Вершинный шейдер

Vertex shader
in Arg 0
in Arg 1
...
in Arg n

gl_Position
out Attr 1
...
out Attr k

```glsl
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;

out vec3 fragPos; // координаты точки
out vec3 fragNorm; // координаты нормали в точке

uniform mat4 clipView;
uniform mat4 modelView;
uniform mat4 modelInv;

void main() {
  fragPos = vec3(modelView * vec4(position, 1.0));
  fragNorm = mat3(modelInv) * normal;
  gl_Position = clipView * vec4(position, 1.0);
}
```

Фрагментный шейдер

Fragment shader

in Attr 1
...
In Attr k

out Color

```
#version 330 core

in vec3 fragPos;
in vec3 fragNorm;

out vec4 color;

uniform vec3 pathColor;
uniform Light light;
uniform vec3 lightPos;
uniform vec3 viewPos;
```

Шейдерная программа

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n

Vertex shader
in Arg 0
in Arg 1
…
in Arg n

$gl\_Position$
out Attr 1
…
out Attr k

Fragment shader

in Attr 1
…
In Attr k

out Color

Шейдерная программа

Loc 0	Loc 1	…	Loc n
A	Attr 1	…	Attr n
B	Attr 1	…	Attr n
C	Attr 1	…	Attr n
D	Attr 1	…	Attr n

Vertex shader
in Arg 0
in Arg 1
…
in Arg n

$gl\_Position$
out Attr 1
…
out Attr k

Fragment shader

in Attr 1
…
In Attr k

out Color

Переменные uniform

Шейдерная программа

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;

out vec3 fragPos; // координаты точки
out vec3 fragNorm; // координаты нормали в точке
```

```
#version 330 core

in vec3 fragPos;
in vec3 fragNorm;

out vec4 color;

uniform vec3 pathColor;
uniform Light light;
uniform vec3 lightPos;
uniform vec3 viewPos;
```

OpenGL pipeline