

Компьютерная графика

Лекция 2: Растеризация, графический конвейер, шейдеры, аффинные преобразования

2021

Два слова о GPU

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data
- ▶ Как SIMD-расширения процессоров (SSE, AVX), но на более крупном масштабе – сотни/тысячи одновременных вычислений

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data
- ▶ Как SIMD-расширения процессоров (SSE, AVX), но на более крупном масштабе – сотни/тысячи одновременных вычислений
- ▶ Для производительности вычислительные ядра организованы в группы (wavefronts) по 32/64 элемента

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data
- ▶ Как SIMD-расширения процессоров (SSE, AVX), но на более крупном масштабе – сотни/тысячи одновременных вычислений
- ▶ Для производительности вычислительные ядра организованы в группы (wavefronts) по 32/64 элемента
- ▶ Все ядра одного wavefront'а выполняют одну и ту же операцию (но над разными данными)

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data
- ▶ Как SIMD-расширения процессоров (SSE, AVX), но на более крупном масштабе – сотни/тысячи одновременных вычислений
- ▶ Для производительности вычислительные ядра организованы в группы (wavefronts) по 32/64 элемента
- ▶ Все ядра одного wavefront'а выполняют одну и ту же операцию (но над разными данными)
- ▶ Если мы рисуем только один треугольник, под него всё равно выделяется целый wavefront и большинство его ядер делают бесполезную работу

Два слова о GPU

- ▶ SIMD – Single Instruction, Multiple Data
- ▶ Как SIMD-расширения процессоров (SSE, AVX), но на более крупном масштабе – сотни/тысячи одновременных вычислений
- ▶ Для производительности вычислительные ядра организованы в группы (wavefronts) по 32/64 элемента
- ▶ Все ядра одного wavefront'а выполняют одну и ту же операцию (но над разными данными)
- ▶ Если мы рисуем только один треугольник, под него всё равно выделяется целый wavefront и большинство его ядер делают бесполезную работу
- ▶ Максимизация загрузки wavefront'ов – важная часть низкоуровневой оптимизации GPU

Растеризация

- ▶ Растеризация – превращение геометрического примитива (точки, линии, треугольника, прямоугольника, круга, и т.д.) в набор соответствующих ему пикселей на экране/изображении
- ▶ Превращение векторных данных в растровые

Растеризация

- ▶ Растеризация – превращение геометрического примитива (точки, линии, треугольника, прямоугольника, круга, и т.д.) в набор соответствующих ему пикселей на экране/изображении
- ▶ Превращение векторных данных в растровые
- ▶ За нас её делает OpenGL!

Растеризация

- ▶ Растеризация – превращение геометрического примитива (точки, линии, треугольника, прямоугольника, круга, и т.д.) в набор соответствующих ему пикселей на экране/изображении
- ▶ Превращение векторных данных в растровые
- ▶ За нас её делает OpenGL!
- ▶ Некоторые современные графические движки GPU (Unreal 5 Nanite) делают растеризацию сами с помощью compute шейдеров

Растрезация: точка

- ▶ Как растрезовать точку (x, y) ?

Растеризация: точка

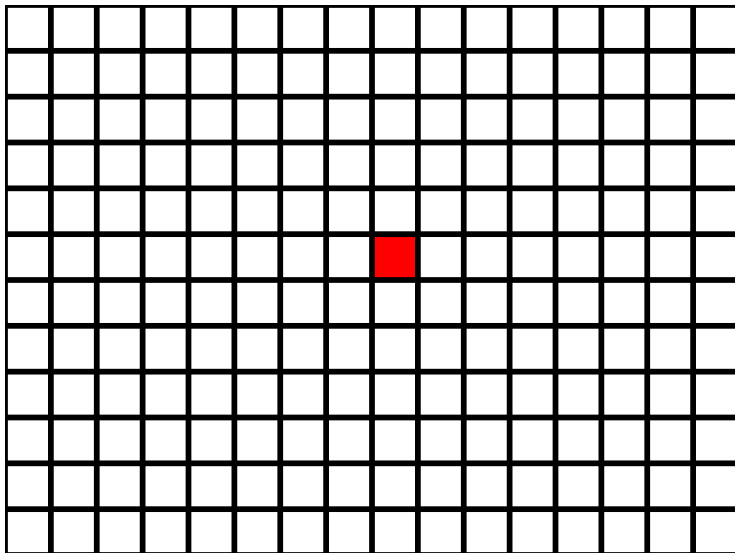
- ▶ Как растеризовать точку (x, y) ?

```
set_pixel(round(x), round(y), color);
```

Растеризация: точка

- ▶ Как растеризовать точку (x, y) ?
`set_pixel(round(x), round(y), color);`
- ▶ В OpenGL: GL_POINTS

Растеризация: точка



Растрезация: линия

- ▶ Как растрезовать линию $(x_1, y_1) \dots (x_2, y_2)$?

Растеризация: линия

- ▶ Как растеризовать линию $(x_1, y_1) \dots (x_2, y_2)$?
- ▶ Алгоритм Брезенхэма

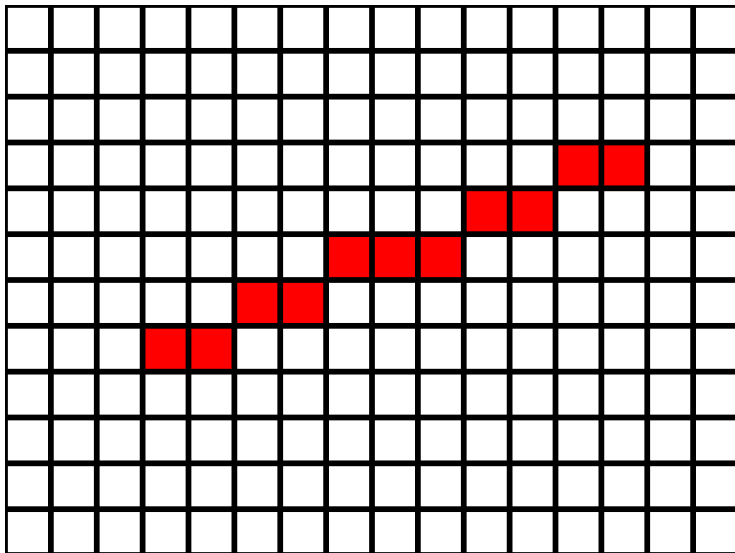
Растрезация: линия

- ▶ Как растрезировать линию $(x_1, y_1) \dots (x_2, y_2)$?
- ▶ Алгоритм Брезенхэма
- ▶ Есть вариация алгоритма для рисования окружностей

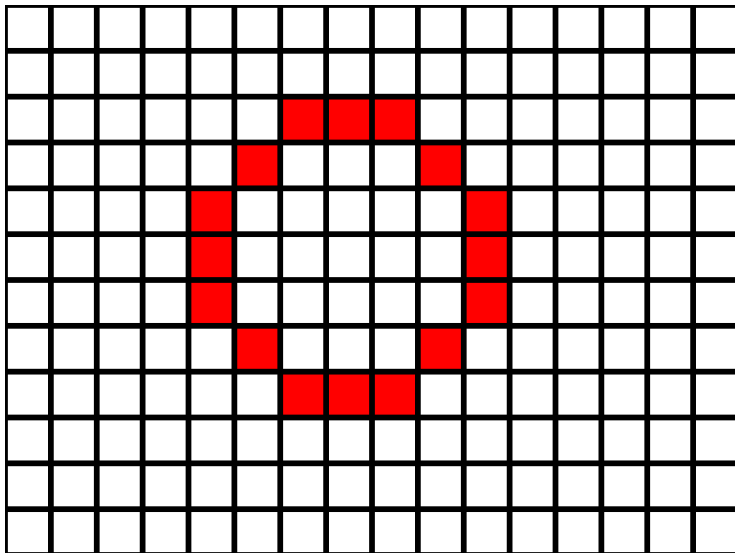
Растризация: линия

- ▶ Как растеризовать линию $(x_1, y_1) \dots (x_2, y_2)$?
- ▶ Алгоритм Брезенхэма
- ▶ Есть вариация алгоритма для рисования окружностей
- ▶ В OpenGL: GL_LINES

Растризация: линия



Растрезация: окружность



Растрезация: прямоугольник

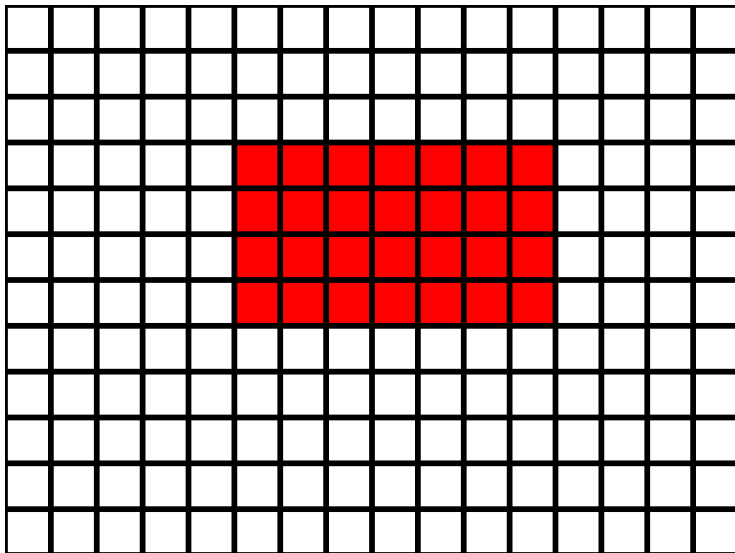
- ▶ Как растрезовать прямоугольник $[x_1 \dots x_2] \times [y_1 \dots y_2]$?

Растрезизация: прямоугольник

- ▶ Как растрезизовать прямоугольник $[x_1 \dots x_2] \times [y_1 \dots y_2]$?

```
for (int x = round(x_1); x <= round(x_2); ++x) {  
    for (int y = round(y_1); y <= round(y_2); ++y) {  
        set_pixel(x, y, color);  
    }  
}
```

Растрезация: прямоугольник



Растеризация: треугольник

- ▶ Как растеризовать треугольник с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$?

Растрезация: треугольник

- ▶ Как растрезировать треугольник с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$?
- ▶ Растрезируем ограничивающий прямоугольник, проверяя пиксели на вхождение в треугольник

Растрезация: треугольник

- ▶ Как растрезировать треугольник с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$?
- ▶ Растрезируем ограничивающий прямоугольник, проверяя пиксели на входжение в треугольник

```
int xmin = min(round(x_1), round(x_2), round(x_3));  
int xmax = max(round(x_1), round(x_2), round(x_3));
```

```
int ymin = min(round(y_1), round(y_2), round(y_3));  
int ymax = max(round(y_1), round(y_2), round(y_3));
```

```
for (int x = xmin; x <= xmax; ++x) {  
    for (int y = ymin; y <= ymax; ++y) {  
        if (inside_triangle(x, y, ...))  
            set_pixel(x, y, color);  
    }  
}
```

Растрезизация: треугольник

- ▶ Как растрезизовать треугольник с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$?
- ▶ Растрезизуем ограничивающий прямоугольник, проверяя пиксели на входжение в треугольник

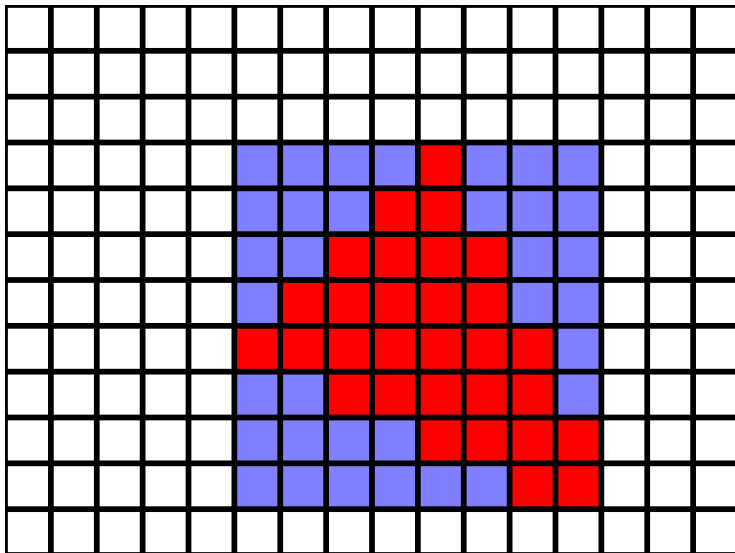
```
int xmin = min(round(x_1), round(x_2), round(x_3));  
int xmax = max(round(x_1), round(x_2), round(x_3));
```

```
int ymin = min(round(y_1), round(y_2), round(y_3));  
int ymax = max(round(y_1), round(y_2), round(y_3));
```

```
for (int x = xmin; x <= xmax; ++x) {  
    for (int y = ymin; y <= ymax; ++y) {  
        if (inside_triangle(x, y, ...))  
            set_pixel(x, y, color);  
    }  
}
```

- ▶ В OpenGL: GL_TRIANGLES

Растеризация: треугольник



Растеризация: круг

- ▶ Как растеризовать круг с центром (x_0, y_0) и радиусом R ?

Растрезация: круг

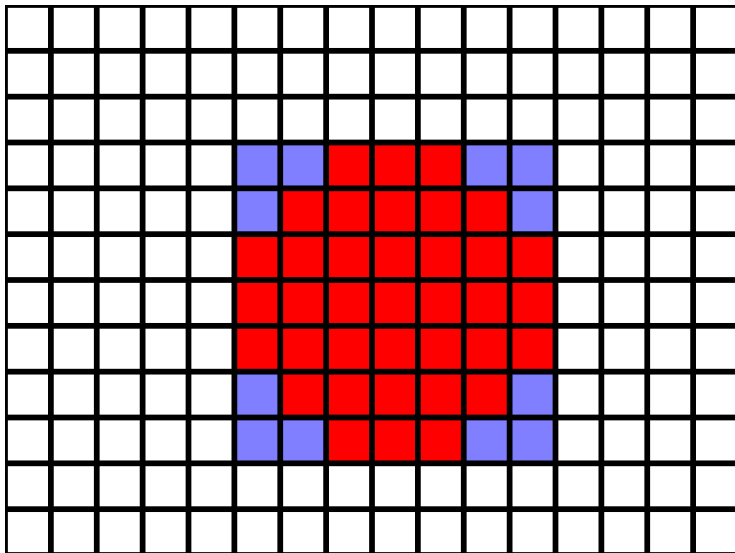
- ▶ Как растрезировать круг с центром (x_0, y_0) и радиусом R ?
- ▶ Растрезируем ограничивающий прямоугольник, проверяя пиксели на входжение в круг

Растрезизация: круг

- ▶ Как растрезизовать круг с центром (x_0, y_0) и радиусом R ?
- ▶ Растрезизуем ограничивающий прямоугольник, проверяя пиксели на входжение в круг

```
int xmin = round(x_0 - R);  
int xmax = round(x_0 + R);  
  
int ymin = round(y_0 - R);  
int ymax = round(y_0 + R);  
  
for (int x = xmin; x <= xmax; ++x) {  
    for (int y = ymin; y <= ymax; ++y) {  
        if (sqr(x - x_0) + sqr(y - y_0) <= sqr(R))  
            set_pixel(x, y, color);  
    }  
}
```

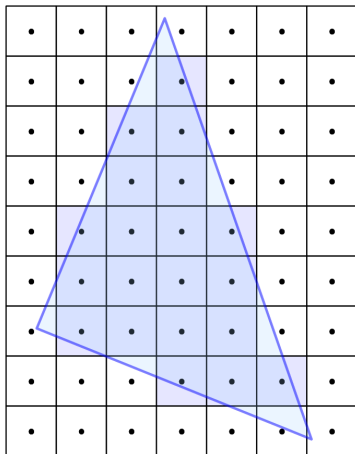

Растеризация: круг



Растеризация: OpenGL

- ▶ Пиксель растеризуется, если центр пикселя содержится в треугольнике

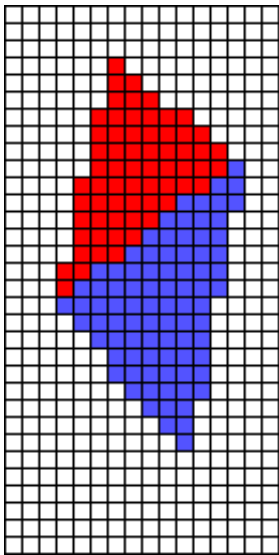
Растеризация: OpenGL



Растеризация: OpenGL

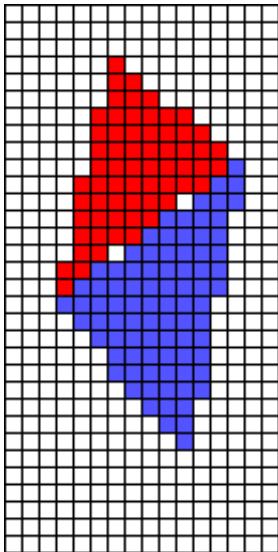
- ▶ Пиксель растеризуется, если центр пикселя содержится в треугольнике
- ▶ Если у двух треугольников есть общее ребро (и они не пересекаются внутренностями), то
 - ▶ Каждый пиксель будет принадлежать ровно одному треугольнику, т.е. не будет наложения
 - ▶ Ни один пиксель не будет пропущен, т.е. не будет “дырок”

Растеризация: OpenGL



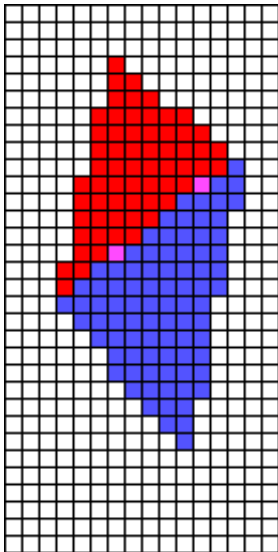
Растеризация: OpenGL

Не будет “дырок”:



Растеризация: OpenGL

Не будет наложения пикселей:



Растеризация: OpenGL

- ▶ Пиксель растеризуется, если центр пикселя содержится в треугольнике
- ▶ Если у двух треугольников есть общее ребро (и они не пересекаются внутренностями), то
 - ▶ Каждый пиксель будет принадлежать ровно одному треугольнику, т.е. не будет наложения
 - ▶ Ни один пиксель не будет пропущен, т.е. не будет “дырок”
- ▶ Подробнее: en.wikibooks.org/wiki/GLSL_Programming/Rasterization

Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:

Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:
- ▶ Точки: `GL_POINTS`

Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:
- ▶ Точки: `GL_POINTS`
- ▶ Линии: `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_LINES`

Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:
- ▶ Точки: `GL_POINTS`
- ▶ Линии: `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_LINES`
- ▶ Треугольники: `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_TRIANGLES`

Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:
- ▶ Точки: `GL_POINTS`
- ▶ Линии: `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_LINES`
- ▶ Треугольники: `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_TRIANGLES`
- ▶ Для геометрических шейдеров:
`GL_LINE_STRIP_ADJACENCY`, `GL_LINES_ADJACENCY`,
`GL_TRIANGLE_STRIP_ADJACENCY`, `GL_TRIANGLES_ADJACENCY`

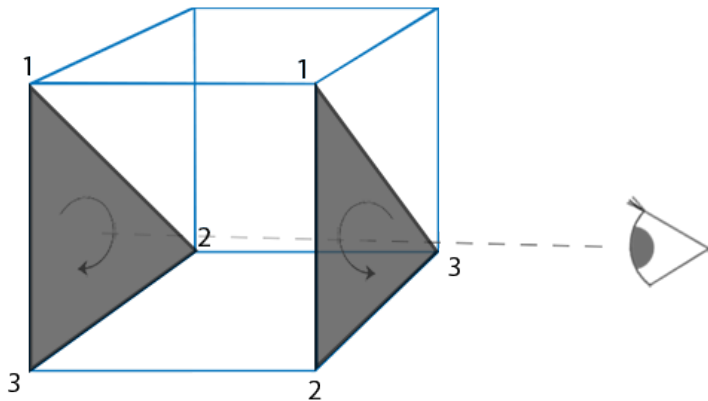
Растеризация: OpenGL

- ▶ В современном OpenGL есть только три примитива для растеризации:
- ▶ Точки: `GL_POINTS`
- ▶ Линии: `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_LINES`
- ▶ Треугольники: `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_TRIANGLES`
- ▶ Для геометрических шейдеров:
`GL_LINE_STRIP_ADJACENCY`, `GL_LINES_ADJACENCY`,
`GL_TRIANGLE_STRIP_ADJACENCY`, `GL_TRIANGLES_ADJACENCY`
- ▶ Точки и линии обычно используются для дебажного рендеринга, главный примитив – треугольник

Back-face culling

- ▶ Обычно в OpenGL треугольники, вершины которых оказываются на экране в порядке обхода **по часовой стрелке**, **не рисуются**

Back-face culling



Back-face culling

- ▶ Обычно в OpenGL треугольники, вершины которых оказываются на экране в порядке обхода **по часовой стрелке**, **не рисуются**
 - ▶ Чтобы не рисовать треугольники, которые всё равно будут скрыты другими треугольниками спереди

Back-face culling

- ▶ Обычно в OpenGL треугольники, вершины которых оказываются на экране в порядке обхода **по часовой стрелке**, **не рисуются**
 - ▶ Чтобы не рисовать треугольники, которые всё равно будут скрыты другими треугольниками спереди
- ▶ Включить/выключить это поведение:
`glEnable(GL_CULL_FACE)` или `glDisable(GL_CULL_FACE)`
(по умолчанию выключено)

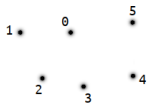
Back-face culling

- ▶ Обычно в OpenGL треугольники, вершины которых оказываются на экране в порядке обхода **по часовой стрелке**, **не рисуются**
 - ▶ Чтобы не рисовать треугольники, которые всё равно будут скрыты другими треугольниками спереди
- ▶ Включить/выключить это поведение:
`glEnable(GL_CULL_FACE)` или `glDisable(GL_CULL_FACE)`
(по умолчанию выключено)
- ▶ Настроить, какие треугольники будут скрываться:
`glCullFace(GL_BACK)`, `glCullFace(GL_FRONT)`,
`glCullFace(GL_FRONT_AND_BACK)`

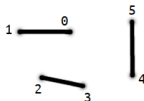
Back-face culling

- ▶ Обычно в OpenGL треугольники, вершины которых оказываются на экране в порядке обхода **по часовой стрелке**, **не рисуются**
 - ▶ Чтобы не рисовать треугольники, которые всё равно будут скрыты другими треугольниками спереди
- ▶ Включить/выключить это поведение:
`glEnable(GL_CULL_FACE)` или `glDisable(GL_CULL_FACE)`
(по умолчанию выключено)
- ▶ Настроить, какие треугольники будут скрываться:
`glCullFace(GL_BACK)`, `glCullFace(GL_FRONT)`,
`glCullFace(GL_FRONT_AND_BACK)`
- ▶ Настроить, какие треугольники считаются FRONT, а какие - BACK: `glFrontFace(GL_CCW)`, `glFrontFace(GL_CW)`

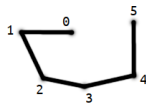
Группировка вершин по примитивам (primitive assembly)



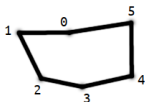
GL_POINTS



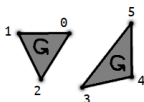
GL_LINES



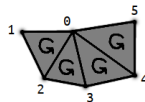
GL_LINE_STRIP



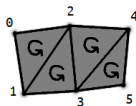
GL_LINE_LOOP



GL_TRIANGLES



GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP

Графический конвейер (graphics pipeline)

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)
- ▶ Преобразование в оконную систему координат (viewport transform)
 - ▶ $X : [-1, 1] \rightarrow [0, width]$
 - ▶ $Y : [-1, 1] \rightarrow [height, 0]$ (-1 внизу, 1 вверху)
 - ▶ `glViewport(0, 0, width, height)`

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)
- ▶ Преобразование в оконную систему координат (viewport transform)
 - ▶ $X : [-1, 1] \rightarrow [0, width]$
 - ▶ $Y : [-1, 1] \rightarrow [height, 0]$ (-1 внизу, 1 вверху)
 - ▶ `glViewport(0, 0, width, height)`
- ▶ Back-face culling

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)
- ▶ Преобразование в оконную систему координат (viewport transform)
 - ▶ $X : [-1, 1] \rightarrow [0, width]$
 - ▶ $Y : [-1, 1] \rightarrow [height, 0]$ (-1 внизу, 1 вверху)
 - ▶ `glViewport(0, 0, width, height)`
- ▶ Back-face culling
- ▶ Растеризация примитивов: примитив превращается в набор пикселей
 - ▶ Линейная интерполяция значений, переданных из вершинного шейдера во фрагментный

Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
 - ▶ Позже мы узнаем, как задавать сами вершины, а не только их количество
- ▶ Вершинный шейдер: обрабатывает вершины по одной
 - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)
- ▶ Преобразование в оконную систему координат (viewport transform)
 - ▶ $X : [-1, 1] \rightarrow [0, width]$
 - ▶ $Y : [-1, 1] \rightarrow [height, 0]$ (-1 внизу, 1 вверху)
 - ▶ `glViewport(0, 0, width, height)`
- ▶ Back-face culling
- ▶ Растеризация примитивов: примитив превращается в набор пикселей
 - ▶ Линейная интерполяция значений, переданных из вершинного шейдера во фрагментный
- ▶ Пиксельный (фрагментный) шейдер: обрабатывает пиксели по одному

Графический конвейер (graphics pipeline)

- ▶ Мы пропустили много важных частей конвейера
- ▶ Будем их по чуть-чуть добавлять в течение курса

Вершинный (vertex) шейдер

Вершинный (vertex) шейдер

- ▶ Входные данные:

Вершинный (vertex) шейдер

- ▶ Входные данные:
 - ▶ Атрибуты вершин (мы позже узнаем, как их задавать) – свои для каждой вершины

Вершинный (vertex) шейдер

- ▶ Входные данные:
 - ▶ Атрибуты вершин (мы позже узнаем, как их задавать) – свои для каждой вершины
 - ▶ Uniform-переменные – глобальные значения, не меняющиеся в течение одного вызова команды рисования (`glDrawArrays`):
`uniform float scale`

Вершинный (vertex) шейдер

- ▶ Входные данные:
 - ▶ Атрибуты вершин (мы позже узнаем, как их задавать) – свои для каждой вершины
 - ▶ Uniform-переменные – глобальные значения, не меняющиеся в течение одного вызова команды рисования (`glDrawArrays`):
`uniform float scale`
- ▶ Выходные данные:
 - ▶ `vec4 gl_Position`

Вершинный (vertex) шейдер

- ▶ Входные данные:
 - ▶ Атрибуты вершин (мы позже узнаем, как их задавать) – свои для каждой вершины
 - ▶ Uniform-переменные – глобальные значения, не меняющиеся в течение одного вызова команды рисования (`glDrawArrays`):
`uniform float scale`
- ▶ Выходные данные:
 - ▶ `vec4 gl_Position`
 - ▶ Переменные, интерполированное значение которых попадёт во фрагментный (пиксельный) шейдер: `out vec3 color`

Фрагментный (пиксельный, fragment) шейдер

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные
 - ▶ Проинтерполированные out-переменные вершинного шейдера: `in vec3 color`

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные
 - ▶ Проинтерполированные out-переменные вершинного шейдера: `in vec3 color`
 - ▶ `gl_FragCoord` – координаты пикселя $(-1 \dots 1)$

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные
 - ▶ Проинтерполированные out-переменные вершинного шейдера: `in vec3 color`
 - ▶ `gl_FragCoord` – координаты пикселя ($-1 \dots 1$)
 - ▶ И много других: khronos.org/opengl/wiki/Fragment_Shader/Defined_Inputs

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные
 - ▶ Проинтерполированные out-переменные вершинного шейдера: `in vec3 color`
 - ▶ `gl_FragCoord` – координаты пикселя ($-1 \dots 1$)
 - ▶ И много других: khronos.org/opengl/wiki/Fragment_Shader/Defined_Inputs
- ▶ Выходные данные:
 - ▶ `layout (location = 0) out vec4 out_color;` – выходной цвет в формате RGBA

Фрагментный (пиксельный, fragment) шейдер

- ▶ Входные данные:
 - ▶ Uniform-переменные
 - ▶ Проинтерполированные out-переменные вершинного шейдера: `in vec3 color`
 - ▶ `gl_FragCoord` – координаты пикселя ($-1 \dots 1$)
 - ▶ И много других: khronos.org/opengl/wiki/Fragment_Shader/Defined_Inputs
- ▶ Выходные данные:
 - ▶ `layout (location = 0) out vec4 out_color;` – выходной цвет в формате RGBA
 - ▶ Может быть несколько, об этом поговорим позже

Языки описания шейдеров

- ▶ OpenGL, Vulkan: GLSL (GL Shading Language)

Языки описания шейдеров

- ▶ OpenGL, Vulkan: GLSL (GL Shading Language)
- ▶ DirectX: HLSL (High-Level Shading Language)

Языки описания шейдеров

- ▶ OpenGL, Vulkan: GLSL (GL Shading Language)
- ▶ DirectX: HLSL (High-Level Shading Language)
- ▶ DurectX (до 2012): Cg (C for Graphics), deprecated

Языки описания шейдеров

- ▶ OpenGL, Vulkan: GLSL (GL Shading Language)
- ▶ DirectX: HLSL (High-Level Shading Language)
- ▶ DurectX (до 2012): Cg (C for Graphics), deprecated
- ▶ en.wikipedia.org/wiki/Shading_language

Язык описания шейдеров GLSL

- ▶ Похож на C

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...
- ▶ Конструкторы векторов:

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...
- ▶ Конструкторы векторов:
 - ▶ `vec3(x, y, z)`

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...
- ▶ Конструкторы векторов:
 - ▶ `vec3(x, y, z)`
 - ▶ `vec3(x) == vec3(x, x, x)`

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...
- ▶ Конструкторы векторов:
 - ▶ `vec3(x, y, z)`
 - ▶ `vec3(x) == vec3(x, x, x)`
 - ▶ `vec3(vec2(x, y), z) == vec3(x, vec2(y, z))`

Язык описания шейдеров GLSL

- ▶ Похож на C
- ▶ Типы данных:
 - ▶ Скалярные: `bool`, `int`, `uint`, `float`
 - ▶ Векторные: `bvec2`, `bvec3`, `bvec4`, `ivec2`, ..., `uvec2`, ..., `vec2`, ...
 - ▶ Матричные: `mat2`, `mat3`, `mat4`, `mat2x4`, ...
 - ▶ В GLSL 400 или с расширением `GL_ARB_gpu_shader_fp64` есть `double`, `dvec2`, ..., `dmat2`, ...
- ▶ Конструкторы векторов:
 - ▶ `vec3(x, y, z)`
 - ▶ `vec3(x) == vec3(x, x, x)`
 - ▶ `vec3(vec2(x, y), z) == vec3(x, vec2(y, z))`
- ▶ Подробнее: [khronos.org/opengl/wiki/Data_Type_\(GLSL\)](http://khronos.org/opengl/wiki/Data_Type_(GLSL))

Язык описания шейдеров GLSL

- ▶ Программа должна начинаться с
`#version <версия> [<профиль>]`

Язык описания шейдеров GLSL

- ▶ Программа должна начинаться с `#version <версия> [<профиль>]`
- ▶ Программа должна содержать функцию `void main()`

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$
- ▶ Swizzling: $a.xxzy == \text{vec4}(a.x, a.x, a.z, a.y)$

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$
- ▶ Swizzling: $a.xxzy == \text{vec4}(a.x, a.x, a.z, a.y)$
- ▶ Доступ к элементам матриц: $m[\text{column}][\text{row}]$

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$
- ▶ Swizzling: $a.xxzy == \text{vec4}(a.x, a.x, a.z, a.y)$
- ▶ Доступ к элементам матриц: $m[\text{column}][\text{row}]$
- ▶ Есть полезные математические функции: pow , \sin , \cos , ...

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$
- ▶ Swizzling: $a.xxzy == \text{vec4}(a.x, a.x, a.z, a.y)$
- ▶ Доступ к элементам матриц: $m[\text{column}][\text{row}]$
- ▶ Есть полезные математические функции: pow , \sin , \cos , ...
 - ▶ Для векторов и матриц применяются покомпонентно

Язык описания шейдеров GLSL

- ▶ Есть стандартные операции: $+$, $-$, $*$, $/$, $\%$, $<$, $==$, ...
 - ▶ Для векторов и матриц применяются покомпонентно
 - ▶ Исключение: умножение двух матриц это умножение в смысле алгебры
 - ▶ Исключение: умножение матрицы на вектор это умножение в смысле алгебры
- ▶ Доступ к координатам векторов: $a.x = b.y$
- ▶ Swizzling: $a.xxzy == \text{vec4}(a.x, a.x, a.z, a.y)$
- ▶ Доступ к элементам матриц: $m[\text{column}][\text{row}]$
- ▶ Есть полезные математические функции: pow , \sin , \cos , ...
 - ▶ Для векторов и матриц применяются покомпонентно
- ▶ Есть математические функции векторной алгебры: dot , cross , length , normalize , ...

Язык описания шейдеров GLSL

- ▶ Есть массивы: `float array[5];`

Язык описания шейдеров GLSL

- ▶ Есть массивы: `float array[5];`
 - ▶ Инициализация:
`float array[5] = float[5](0.0, 1.0, 2.0, 3.0, 4.0);`

Язык описания шейдеров GLSL

- ▶ Есть массивы: `float array[5];`
 - ▶ Инициализация:
`float array[5] = float[5](0.0, 1.0, 2.0, 3.0, 4.0);`
- ▶ Константы (известные на момент компиляции):
`const float PI = 3.141592;`

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`
- ▶ Тернарный if: `condition ? x : y`

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`
- ▶ Тернарный if: `condition ? x : y`
- ▶ Циклы: `for (int i = 0; i < 10; ++i) { ... }`
 - ▶ Число итераций цикла должно быть константой, известной на момент компиляции шейдера
 - ▶ **Не может зависеть** от uniform-переменных, атрибутов вершин, и т.д.

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`
- ▶ Тернарный if: `condition ? x : y`
- ▶ Циклы: `for (int i = 0; i < 10; ++i) { ... }`
 - ▶ Число итераций цикла должно быть константой, известной на момент компиляции шейдера
 - ▶ **Не может зависеть** от uniform-переменных, атрибутов вершин, и т.д.

- ▶ Функции:

```
vec3 reflect(vec3 v, vec3 n) {  
    return v - 2.0 * n * dot(v, n);  
}
```

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`
- ▶ Тернарный if: `condition ? x : y`
- ▶ Циклы: `for (int i = 0; i < 10; ++i) { ... }`
 - ▶ Число итераций цикла должно быть константой, известной на момент компиляции шейдера
 - ▶ **Не может зависеть** от uniform-переменных, атрибутов вершин, и т.д.

- ▶ Функции:

```
vec3 reflect(vec3 v, vec3 n) {  
    return v - 2.0 * n * dot(v, n);  
}
```

- ▶ Могут вызывать другие функции

Язык описания шейдеров GLSL

- ▶ Ветвление: `if (condition) { ... } else { ... }`
- ▶ Тернарный if: `condition ? x : y`
- ▶ Циклы: `for (int i = 0; i < 10; ++i) { ... }`
 - ▶ Число итераций цикла должно быть константой, известной на момент компиляции шейдера
 - ▶ **Не может зависеть** от uniform-переменных, атрибутов вершин, и т.д.

- ▶ Функции:

```
vec3 reflect(vec3 v, vec3 n) {  
    return v - 2.0 * n * dot(v, n);  
}
```

- ▶ Могут вызывать другие функции
- ▶ Рекурсия **запрещена**

Полезные ресурсы о шейдерах

- ▶ Тьюториал: learnopengl.com/Getting-started/Shader
- ▶ Тьюториал: lighthouse3d.com/tutorials/glsl-tutorial

Полезные ресурсы о шейдерах

- ▶ Тьюториал: learnopengl.com/Getting-started/Shader
- ▶ Тьюториал: lighthouse3d.com/tutorials/glsl-tutorial
- ▶ Книжка-учебник с большим количеством примеров сложных шейдеров: The Book of Shaders

Полезные ресурсы о шейдерах

- ▶ Тьюториал: learnopengl.com/Getting-started/Shader
- ▶ Тьюториал: lighthouse3d.com/tutorials/glsl-tutorial
- ▶ Книжка-учебник с большим количеством примеров сложных шейдеров: The Book of Shaders
- ▶ Онлайн-редактор шейдеров: shadertoy.com

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты
- ▶ Мы хотим рисовать двухмерные и трёхмерные объекты, зачем четвёртая координата?

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты
- ▶ Мы хотим рисовать двухмерные и трёхмерные объекты, зачем четвёртая координата?
- ▶ Три причины:

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты
- ▶ Мы хотим рисовать двухмерные и трёхмерные объекты, зачем четвёртая координата?
- ▶ Три причины:
 - ▶ Для некоторых типов данных: кватернионы, цвета (rgba)

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты
- ▶ Мы хотим рисовать двухмерные и трёхмерные объекты, зачем четвёртая координата?
- ▶ Три причины:
 - ▶ Для некоторых типов данных: кватернионы, цвета (rgba)
 - ▶ Для перспективной проекции (следующая лекция)

Четырёхмерные векторы?

- ▶ GLSL поддерживает операции с матрицами и векторами размерностей вплоть до 4
- ▶ `gl_Position` имеет 4 компоненты
- ▶ Мы хотим рисовать двухмерные и трёхмерные объекты, зачем четвёртая координата?
- ▶ Три причины:
 - ▶ Для некоторых типов данных: кватернионы, цвета (rgba)
 - ▶ Для перспективной проекции (следующая лекция)
 - ▶ Для аффинных преобразований

Аффинное пространство

- ▶ Векторные (линейные) пространства – круто: большая теория, эффективные операции

Аффинное пространство

- ▶ Векторные (линейные) пространства – круто: большая теория, эффективные операции
- ▶ Не очень подходит для того, чтобы моделировать пространство

Аффинное пространство

- ▶ Векторные (линейные) пространства – круто: большая теория, эффективные операции
- ▶ Не очень подходит для того, чтобы моделировать пространство
 - ▶ Где в пространстве точка, соответствующая нулевому вектору?

Аффинное пространство

- ▶ Векторные (линейные) пространства – круто: большая теория, эффективные операции
- ▶ Не очень подходит для того, чтобы моделировать пространство
 - ▶ Где в пространстве точка, соответствующая нулевому вектору?
 - ▶ Что такое сумма двух точек?

Аффинное пространство

- ▶ Векторные (линейные) пространства – круто: большая теория, эффективные операции
- ▶ Не очень подходит для того, чтобы моделировать пространство
 - ▶ Где в пространстве точка, соответствующая нулевому вектору?
 - ▶ Что такое сумма двух точек?
- ▶ Мысль: точки \neq векторы

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:
 - ▶ Множество точек \mathbb{A}

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:
 - ▶ Множество точек \mathbb{A}
 - ▶ Операция “точка + вектор”: $\oplus : \mathbb{A} \times \mathbb{V} \rightarrow \mathbb{A}$

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:
 - ▶ Множество точек \mathbb{A}
 - ▶ Операция “точка + вектор”: $\oplus : \mathbb{A} \times \mathbb{V} \rightarrow \mathbb{A}$
 - ▶ Операция “точка - точка”: $\ominus : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{V}$

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:
 - ▶ Множество точек \mathbb{A}
 - ▶ Операция “точка + вектор”: $\oplus : \mathbb{A} \times \mathbb{V} \rightarrow \mathbb{A}$
 - ▶ Операция “точка - точка”: $\ominus : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{V}$
 - ▶ Аксиомы, связывающие эти аксиомы с операциями векторного пространства, например
$$(p \oplus v_1) \oplus v_2 = p \oplus (v_1 + v_2)$$

Аффинное пространство

- ▶ Аффинное пространство: векторное пространство, в котором “забыли” нулевой вектор
- ▶ Формальнее: аффинное пространство \mathbb{A} над векторным пространством \mathbb{V} – это:
 - ▶ Множество точек \mathbb{A}
 - ▶ Операция “точка + вектор”: $\oplus : \mathbb{A} \times \mathbb{V} \rightarrow \mathbb{A}$
 - ▶ Операция “точка - точка”: $\ominus : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{V}$
 - ▶ Аксиомы, связывающие эти аксиомы с операциями векторного пространства, например
$$(p \oplus v_1) \oplus v_2 = p \oplus (v_1 + v_2)$$
- ▶ Любую точку $p_0 \in \mathbb{A}$ можно принять за начало координат
- ▶ Это даёт биекцию $\mathbb{A} \rightarrow \mathbb{V}$ по формуле $p \mapsto p - p_0$

Аффинное пространство

- ▶ Множество решений уравнения $Lx = 0$ в векторном пространстве – векторное подпространство $\ker L$

Аффинное пространство

- ▶ Множество решений уравнения $Lx = 0$ в векторном пространстве – векторное подпространство $\ker L$
- ▶ Множество решений уравнения $Lx = y$ в векторном пространстве – **не** векторное подпространство

Аффинное пространство

- ▶ Множество решений уравнения $Lx = 0$ в векторном пространстве – векторное подпространство $\ker L$
- ▶ Множество решений уравнения $Lx = y$ в векторном пространстве – **не** векторное подпространство
- ▶ Множество решений уравнения $Lx = y$ в векторном пространстве – аффинное пространство над $\ker L$

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$
- ▶ Теорема: если $\sum \lambda_i = 1$, то результат не зависит от выбора p_0

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$
- ▶ Теорема: если $\sum \lambda_i = 1$, то результат не зависит от выбора p_0
- ▶ В этом случае $q = \sum \lambda_i p_i$ называется аффинной комбинацией точек p_i с коэффициентами λ_i

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$
- ▶ Теорема: если $\sum \lambda_i = 1$, то результат не зависит от выбора p_0
- ▶ В этом случае $q = \sum \lambda_i p_i$ называется аффинной комбинацией точек p_i с коэффициентами λ_i
- ▶ Коэффициенты λ_i называются барицентрическими координатами точки q

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$
- ▶ Теорема: если $\sum \lambda_i = 1$, то результат не зависит от выбора p_0
- ▶ В этом случае $q = \sum \lambda_i p_i$ называется аффинной комбинацией точек p_i с коэффициентами λ_i
- ▶ Коэффициенты λ_i называются барицентрическими координатами точки q
- ▶ Поиск λ_i по известным q и p_i сводится к решению линейной системы уравнений (квадратная только если $N = D + 1$, где D – размерность пространства)

Аффинные комбинации

- ▶ Пусть есть набор точек p_1, \dots, p_N и набор коэффициентов $\lambda_1, \dots, \lambda_N$
- ▶ Выберем начало координат p_0 и определим
$$\sum \lambda_i p_i = p_0 + \sum \lambda_i (p_i - p_0)$$
- ▶ Теорема: если $\sum \lambda_i = 1$, то результат не зависит от выбора p_0
- ▶ В этом случае $q = \sum \lambda_i p_i$ называется аффинной комбинацией точек p_i с коэффициентами λ_i
- ▶ Коэффициенты λ_i называются барицентрическими координатами точки q
- ▶ Поиск λ_i по известным q и p_i сводится к решению линейной системы уравнений (квадратная только если $N = D + 1$, где D – размерность пространства)
- ▶ Аффинная комбинация пустого множества точек не определена

Аффинные комбинации: пример



Аффинные комбинации: пример

$$\lambda_1 = 1$$

$$\lambda_2 = 0$$



p_1



p_2

Аффинные комбинации: пример

$$\lambda_1 = 0$$

$$\lambda_2 = 1$$



p_1



p_2

Аффинные комбинации: пример

$$\lambda_1 = 0.5$$

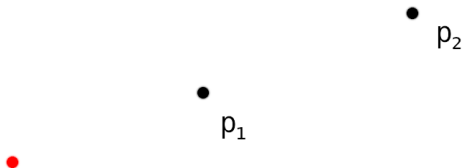
$$\lambda_2 = 0.5$$



Аффинные комбинации: пример

$$\lambda_1 = 2$$

$$\lambda_2 = -1$$



Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N

Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N
- ▶ Их аффинная оболочка – множество всех возможных аффинных комбинаций, т.е. множество $\sum \lambda_i p_i$ для всех возможных λ_i таких, что $\sum \lambda_i = 1$

Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N
- ▶ Их аффинная оболочка – множество всех возможных аффинных комбинаций, т.е. множество $\sum \lambda_i p_i$ для всех возможных λ_i таких, что $\sum \lambda_i = 1$
- ▶ Аффинная оболочка точки – сама точка

Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N
- ▶ Их аффинная оболочка – множество всех возможных аффинных комбинаций, т.е. множество $\sum \lambda_i p_i$ для всех возможных λ_i таких, что $\sum \lambda_i = 1$
- ▶ Аффинная оболочка точки – сама точка
- ▶ Аффинная оболочка двух точек – содержащая их прямая

Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N
- ▶ Их аффинная оболочка – множество всех возможных аффинных комбинаций, т.е. множество $\sum \lambda_i p_i$ для всех возможных λ_i таких, что $\sum \lambda_i = 1$
- ▶ Аффинная оболочка точки – сама точка
- ▶ Аффинная оболочка двух точек – содержащая их прямая
- ▶ Аффинная оболочка трёх точек – содержащая их плоскость

Аффинная оболочка

- ▶ Пусть есть набор точек p_1, \dots, p_N
- ▶ Их аффинная оболочка – множество всех возможных аффинных комбинаций, т.е. множество $\sum \lambda_i p_i$ для всех возможных λ_i таких, что $\sum \lambda_i = 1$
- ▶ Аффинная оболочка точки – сама точка
- ▶ Аффинная оболочка двух точек – содержащая их прямая
- ▶ Аффинная оболочка трёх точек – содержащая их плоскость
- ▶ И т.д.

Линейная интерполяция

- ▶ Линейная интерполяция = интерполяция, использующая барицентрические координаты как коэффициенты

Линейная интерполяция

- ▶ Линейная интерполяция = интерполяция, использующая барицентрические координаты как коэффициенты
- ▶ Пусть есть точки p_i и точка $q = \sum \lambda_i p_i$

Линейная интерполяция

- ▶ Линейная интерполяция = интерполяция, использующая барицентрические координаты как коэффициенты
- ▶ Пусть есть точки p_i и точка $q = \sum \lambda_i p_i$
- ▶ Пусть в точках p_i задано значение некоторой величины f_i

Линейная интерполяция

- ▶ Линейная интерполяция = интерполяция, использующая барицентрические координаты как коэффициенты
- ▶ Пусть есть точки p_i и точка $q = \sum \lambda_i p_i$
- ▶ Пусть в точках p_i задано значение некоторой величины f_i
- ▶ Линейная интерполяция величины f в точке q – значение $\sum \lambda_i f_i$

Линейная интерполяция

- ▶ Линейная интерполяция = интерполяция, использующая барицентрические координаты как коэффициенты
- ▶ Пусть есть точки p_i и точка $q = \sum \lambda_i p_i$
- ▶ Пусть в точках p_i задано значение некоторой величины f_i
- ▶ Линейная интерполяция величины f в точке q – значение $\sum \lambda_i f_i$
- ▶ **Ровно это** происходит при интерполяции значений, переданных из вершинного шейдера во фрагментный

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$
- ▶ Множество всех выпуклых комбинаций набора точек = выпуклая оболочка

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$
- ▶ Множество всех выпуклых комбинаций набора точек = выпуклая оболочка
- ▶ Выпуклая оболочка точки – сама точка

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$
- ▶ Множество всех выпуклых комбинаций набора точек = выпуклая оболочка
- ▶ Выпуклая оболочка точки – сама точка
- ▶ Выпуклая оболочка двух точек – отрезок между этими точками

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$
- ▶ Множество всех выпуклых комбинаций набора точек = выпуклая оболочка
- ▶ Выпуклая оболочка точки – сама точка
- ▶ Выпуклая оболочка двух точек – отрезок между этими точками
- ▶ Выпуклая оболочка трёх точек – треугольник с этими точками в качестве вершин

Выпуклые оболочки

- ▶ Выпуклая комбинация: аффинная комбинация, в которой все коэффициенты неотрицательны $\lambda_i \geq 0$
- ▶ Множество всех выпуклых комбинаций набора точек = выпуклая оболочка
- ▶ Выпуклая оболочка точки – сама точка
- ▶ Выпуклая оболочка двух точек – отрезок между этими точками
- ▶ Выпуклая оболочка трёх точек – треугольник с этими точками в качестве вершин
- ▶ Выпуклая оболочка четырёх точек – тетраэдр с этими точками в качестве вершин

Аффинные преобразования

- ▶ Линейные преобразования – сохраняют линейные комбинации векторов: $S(\sum \lambda_i v_i) = \sum \lambda_i S v_i$

Аффинные преобразования

- ▶ Линейные преобразования – сохраняют линейные комбинации векторов: $S(\sum \lambda_i v_i) = \sum \lambda_i S v_i$
- ▶ Аффинные преобразования – сохраняют аффинные комбинации точек: $S(\sum \lambda_i p_i) = \sum \lambda_i S p_i$

Аффинные преобразования

- ▶ Линейные преобразования – сохраняют линейные комбинации векторов: $S(\sum \lambda_i v_i) = \sum \lambda_i S v_i$
- ▶ Аффинные преобразования – сохраняют аффинные комбинации точек: $S(\sum \lambda_i p_i) = \sum \lambda_i S p_i$
- ▶ При выборе начала координат p_0 преобразование $S : \mathbb{A} \rightarrow \mathbb{A}$ можно понимать как преобразование соответствующего векторного пространства $S : \mathbb{V} \rightarrow \mathbb{V}$ по формуле $S(v) = S(p_0 + v) - p_0$

Аффинные преобразования

- ▶ Линейные преобразования – сохраняют линейные комбинации векторов: $S(\sum \lambda_i v_i) = \sum \lambda_i S v_i$
- ▶ Аффинные преобразования – сохраняют аффинные комбинации точек: $S(\sum \lambda_i p_i) = \sum \lambda_i S p_i$
- ▶ При выборе начала координат p_0 преобразование $S : \mathbb{A} \rightarrow \mathbb{A}$ можно понимать как преобразование соответствующего векторного пространства $S : \mathbb{V} \rightarrow \mathbb{V}$ по формуле $S(v) = S(p_0 + v) - p_0$
- ▶ Можно показать, что в таком виде любое аффинное преобразование это линейное преобразование + сдвиг на фиксированный вектор:
 $S(v) = Av + b$ где
 - ▶ A - линейное преобразование пространства \mathbb{V}
 - ▶ b - вектор из пространства \mathbb{V}

Аффинные преобразования

- ▶ В коде аффинное преобразование пространства размерности N можно представлять как пару (A, b) из матрицы $N \times N$ и вектора размерности N

Аффинные преобразования

- ▶ В коде аффинное преобразование пространства размерности N можно представлять как пару (A, b) из матрицы $N \times N$ и вектора размерности N
- ▶ Пара (A, b) действует на точку/вектор v по формуле $(A, b) \cdot v = Av + b$

Аффинные преобразования

- ▶ В коде аффинное преобразование пространства размерности N можно представлять как пару (A, b) из матрицы $N \times N$ и вектора размерности N
- ▶ Пара (A, b) действует на точку/вектор v по формуле $(A, b) \cdot v = Av + b$
- ▶ Композиция преобразований:
$$(A_1, b_1) \cdot (A_2, b_2) \cdot v = (A_1, b_1) \cdot (A_2 v + b_2) = A_1(A_2 v + b_2) + b_1 = (A_1 A_2)v + (A_1 b_2 + b_1) = (A_1 A_2, A_1 b_2 + b_1) \cdot v$$

Аффинные преобразования

- ▶ В коде аффинное преобразование пространства размерности N можно представлять как пару (A, b) из матрицы $N \times N$ и вектора размерности N
- ▶ Пара (A, b) действует на точку/вектор v по формуле $(A, b) \cdot v = Av + b$
- ▶ Композиция преобразований:
$$(A_1, b_1) \cdot (A_2, b_2) \cdot v = (A_1, b_1) \cdot (A_2 v + b_2) = A_1(A_2 v + b_2) + b_1 = (A_1 A_2)v + (A_1 b_2 + b_1) = (A_1 A_2, A_1 b_2 + b_1) \cdot v$$
- ▶ Тожественное преобразование: $(\mathbb{I}, 0)$

Аффинные преобразования

- ▶ В коде аффинное преобразование пространства размерности N можно представлять как пару (A, b) из матрицы $N \times N$ и вектора размерности N
- ▶ Пара (A, b) действует на точку/вектор v по формуле $(A, b) \cdot v = Av + b$
- ▶ Композиция преобразований:
$$(A_1, b_1) \cdot (A_2, b_2) \cdot v = (A_1, b_1) \cdot (A_2 v + b_2) = A_1(A_2 v + b_2) + b_1 = (A_1 A_2) v + (A_1 b_2 + b_1) = (A_1 A_2, A_1 b_2 + b_1) \cdot v$$
- ▶ Тожественное преобразование: $(I, 0)$
- ▶ Обратное преобразование:
$$(A, b)^{-1} = (A^{-1}, -A^{-1}b)$$

Аффинные преобразования

- ▶ Повсеместно используются для задания положения, ориентации и размера объектов

Аффинные преобразования

- ▶ Повсеместно используются для задания положения, ориентации и размера объектов
- ▶ Векторная часть преобразования: положение 3D-объекта

Аффинные преобразования

- ▶ Повсеместно используются для задания положения, ориентации и размера объектов
- ▶ Векторная часть преобразования: положение 3D-объекта
- ▶ Матричная часть преобразования: вращение и размер объекта

Аффинные преобразования

- ▶ Повсеместно используются для задания положения, ориентации и размера объектов
- ▶ Векторная часть преобразования: положение 3D-объекта
- ▶ Матричная часть преобразования: вращение и размер объекта
- ▶ При чём тут четвёртая координата?

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты
- ▶ Сoglасуется с операциями на векторах и точках

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты
- ▶ Согласуется с операциями на векторах и точках
- ▶ Согласуется с аффинными комбинациями

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты
- ▶ Сogласуется с операциями на векторах и точках
- ▶ Сogласуется с аффинными комбинациями
- ▶ Позволяет представить аффинное преобразование N -мерного пространства как матрицу $(N + 1) \times (N + 1)$:

$$\begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix}$$

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты
- ▶ Сoglасуется с операциями на векторах и точках
- ▶ Сoglасуется с аффинными комбинациями
- ▶ Позволяет представить аффинное преобразование N -мерного пространства как матрицу $(N + 1) \times (N + 1)$:

$$\begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix}$$

- ▶ Можно применять как к точкам, так и к векторам (векторы игнорируют сдвиг на b)

Однородные координаты

- ▶ Трюк: можно вложить вектор размерности N в векторное пространство размерности $N + 1$, добавив 0 в качестве последней координаты
- ▶ Трюк: можно вложить точку размерности N в векторное пространство размерности $N + 1$, добавив 1 в качестве последней координаты
- ▶ Сoglасуется с операциями на векторах и точках
- ▶ Сoglасуется с аффинными комбинациями
- ▶ Позволяет представить аффинное преобразование N -мерного пространства как матрицу $(N + 1) \times (N + 1)$:

$$\begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix}$$

- ▶ Можно применять как к точкам, так и к векторам (векторы игнорируют сдвиг на b)
- ▶ Композиция преобразований = умножение матриц

Точка в однородных координатах

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix}$$

Вектор в однородных координатах

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$$

Аффинное преобразование в однородных координатах

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & b_1 \\ A_{2,1} & A_{2,2} & A_{2,3} & b_2 \\ A_{3,1} & A_{3,2} & A_{3,3} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Матрица сдвига на фиксированный вектор

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$$

Матрица масштабирования на константу

$$\begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} sp_x \\ sp_y \\ sp_z \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} sv_x \\ sv_y \\ sv_z \\ 0 \end{pmatrix}$$

Матрица поворота на угол в плоскости XY

$$\begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \cos \theta - p_y \sin \theta \\ p_x \sin \theta + p_y \cos \theta \\ p_z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \cos \theta - v_y \sin \theta \\ v_x \sin \theta + v_y \cos \theta \\ v_z \\ 0 \end{pmatrix}$$

Аффинные преобразования: ссылки

- ▶ learnopengl.com/Getting-started/Transformations
- ▶ open.gl/transformations
- ▶ en.wikipedia.org/wiki/Affine_space
- ▶ en.wikipedia.org/wiki/Affine_transformation