

# Компьютерная графика

## Лекция 2: Графический конвейер, проекции, аффинные преобразования

2021

# 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

# 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

# Растеризация: линия

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

# Растеризация: 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

# Растрезация: прямоугольник

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

# Растрезация: прямоугольник

- ▶ Как растрезовать 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: треугольник

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

## Растрезация: треугольник

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

## 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);  
    }  
}
```



## Растеризация: треугольник

- ▶ Как растеризовать треугольник с вершинами  $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ ?
- ▶ Растеризуем ограничивающий прямоугольник, проверяя пиксели на вхождение 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: круг

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

## Растеризация: круг

- ▶ Как растеризовать круг с центром  $(x_0, y_0)$  и радиусом  $R$ ?
- ▶ Растеризуем ограничивающий прямоугольник, проверяя пиксели на вхождение в круг

## Растрезация: круг

- ▶ Как растрезировать круг с центром  $(x_0, y_0)$  и радиусом  $R$ ?
- ▶ Растрезируем 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: 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`



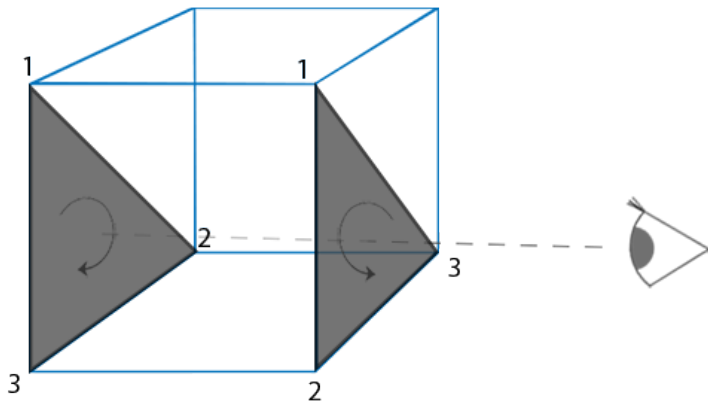
# Растеризация: OpenGL

- ▶ 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`

# Back-face culling

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

# Back-face culling



# Back-face culling

- ▶ По умолчанию 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

- ▶ По умолчанию 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)`

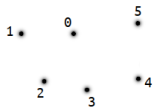
# Back-face culling

- ▶ По умолчанию 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)`
- ▶ Настроить, какие треугольники будут скрываться:  
`glCullFace(GL_BACK)`, `glCullFace(GL_FRONT)`,  
`glCullFace(GL_FRONT_AND_BACK)`

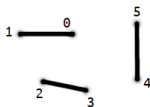
# Back-face culling

- ▶ По умолчанию 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)`
- ▶ 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)`

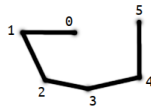
# Группировка вершин по примитивам (primitive assembly)



GL\_POINTS



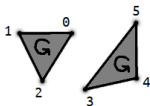
GL\_LINES



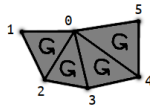
GL\_LINE\_STRIP



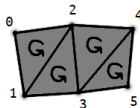
GL\_LINE\_LOOP



GL\_TRIANGLES



GL\_TRIANGLE\_FAN



GL\_TRIANGLE\_STRIP



# Графический конвейер (graphics pipeline)

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
- ▶ Вершинный шейдер: обрабатывает вершины по одной
  - ▶ Должен записать `vec4 gl_Position`

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
- ▶ Вершинный шейдер: обрабатывает вершины по одной
  - ▶ Должен записать `vec4 gl_Position`
- ▶ Сборка примитивов (primitive assembly)

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
- ▶ 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)
  - ▶ Остальные две координаты пока пропустим

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
- ▶ 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)
  - ▶ Остальные две координаты пока пропустим
- ▶ Back-face culling

# Графический конвейер (graphics pipeline)

- ▶ Входной поток вершин (vertex stream)
- ▶ 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)
  - ▶ Остальные две координаты пока пропустим
- ▶ 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)
- ▶ 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)
  - ▶ Остальные две координаты пока пропустим
- ▶ 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