Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Наименование дисциплины: **Алгоритмы Компьютерной Графики**

О Т Ч Е Т

по лабораторной работе 2

Выполнил: **Маргиев Давид, P3369**

Преподаватель: **Андреев Артем Станиславович**

Дата: 13.11.2025

г. Санкт-Петербург
2025

Задание 2:

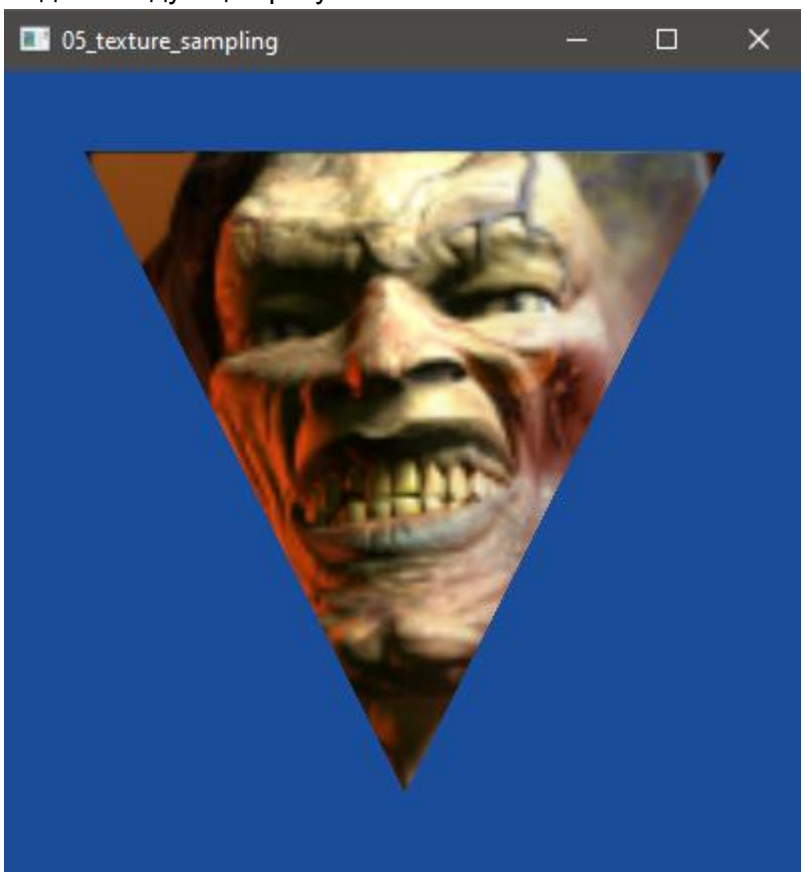
Показать в отчете passthrough координаты в нормализованной системе координат, помним, что деление на w осуществляет ускоритель сам. И показать, реализовав 0.5*xy + vec2(0.5) как они пробегают все значения на экране

1. Выводим UV координаты
2. Выводим текстуру на экран, обязательно свою, а не всяких «чертей» из директории, сопровождающих NV примеры.

Выполнение:

Сначала запустим сэмпл (в директории сэмплов CgToolkit’a обычно он называется Texture Mapping).

Видим следующий результат:



Текстура, которая рендерится на треугольнике - это изображение, лежащее в директории сэмпла - `demon_image.h`
Оно хранит RGB8 изображение размерности 128x128 в виде mipmap-цепи.

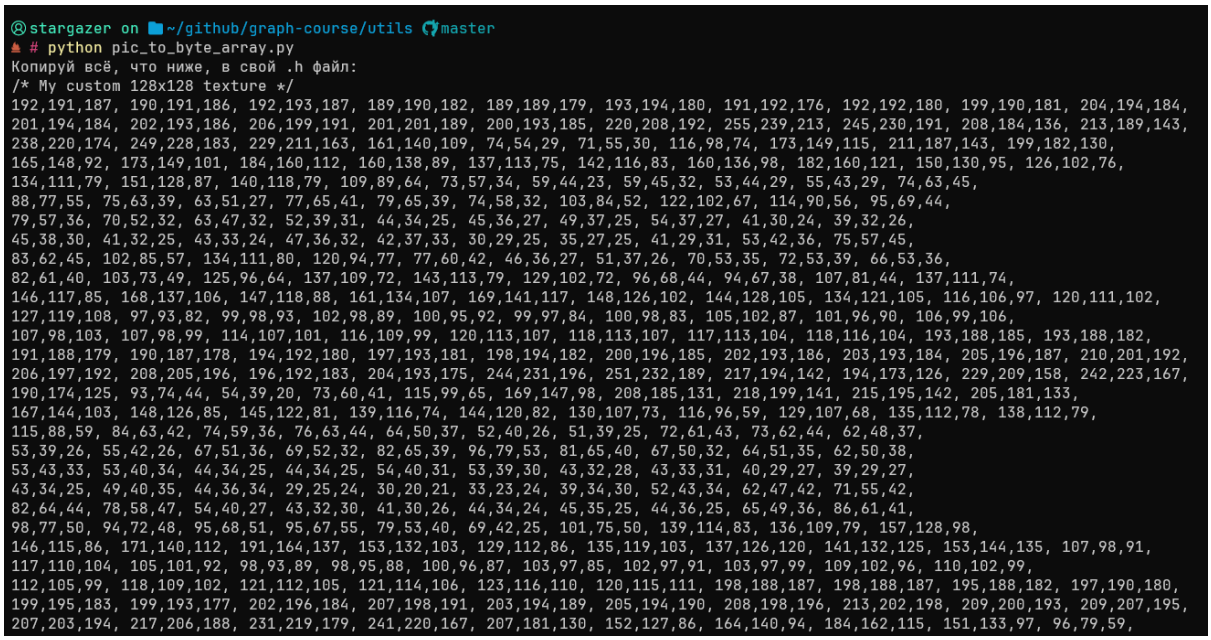
Соответственно, по заданию либо меняем mipmap-цепь, либо делаем новую (или просто используем byte array).

Я решил сделать свою, из следующего изображения, которое я сжал до 128x128:

Для этого, я написал следующий python-скрипт, преобразующий изображение в byte-array:

```
from PIL import Image

img = Image.open('my_image.jpg').convert('RGB')
img = img.resize((128, 128))

pixels = list(img.getdata())
output = []

print("Копируй всё, что ниже, в свой .h файл:")
print("/* My custom 128x128 texture */")

count = 0
line = ""
for r, g, b in pixels:
    line += f"{r},{g},{b}, "
    count += 1
    if count % 10 == 0:
        print(line)
        line = ""

if line:
    print(line)
```

Прогнав изображение через скрипт, я получил byte-array:



```
stargazer on ~/github/graph-course/utils master
# python pic_to_byte_array.py
Копируй всё, что ниже, в свой .h файл:
/* My custom 128x128 texture */
192,191,187, 190,191,186, 192,193,187, 189,190,182, 189,189,179, 193,194,180, 191,192,176, 192,192,180, 199,190,181, 204,194,184,
201,194,184, 202,193,186, 206,199,191, 201,201,189, 200,193,185, 220,208,192, 255,239,213, 245,230,191, 208,184,136, 213,189,143,
238,220,174, 249,228,183, 229,211,163, 161,140,109, 74,54,29, 71,55,30, 116,98,74, 173,149,115, 211,187,143, 199,182,130,
165,148,92, 173,149,101, 184,160,112, 160,138,89, 137,113,75, 142,116,83, 160,136,98, 182,160,121, 150,130,95, 126,102,76,
134,111,79, 151,128,87, 140,118,79, 109,89,64, 73,57,34, 59,44,23, 59,45,32, 53,44,29, 55,43,29, 74,63,45,
88,77,55, 75,63,39, 63,51,27, 77,65,41, 79,65,39, 74,58,32, 103,84,52, 122,102,67, 114,90,56, 95,69,44,
79,57,36, 70,52,32, 63,47,32, 52,39,31, 44,34,25, 45,36,27, 49,37,25, 54,37,27, 41,30,24, 39,32,26,
45,38,30, 41,32,25, 43,33,24, 47,36,32, 42,37,33, 30,29,25, 35,27,25, 41,29,31, 53,42,36, 75,57,45,
83,62,45, 102,85,57, 134,111,80, 120,94,77, 77,60,42, 46,36,27, 51,37,26, 70,53,35, 72,53,39, 66,53,36,
82,61,40, 103,73,49, 125,96,64, 137,109,72, 143,113,79, 129,102,72, 96,68,44, 94,67,38, 107,81,44, 137,111,74,
146,117,85, 168,137,106, 147,118,88, 161,134,107, 169,141,117, 148,126,102, 144,128,105, 134,121,105, 116,106,97, 120,111,102,
127,119,108, 97,93,82, 99,98,93, 102,98,89, 100,95,92, 99,97,84, 100,98,83, 105,102,87, 101,96,90, 106,99,106,
107,98,103, 107,98,99, 114,107,101, 116,109,99, 120,113,107, 118,113,107, 117,113,104, 118,116,104, 193,188,185, 193,188,182,
191,188,179, 190,187,178, 194,192,180, 197,193,181, 198,194,182, 200,196,185, 202,193,186, 203,193,184, 205,196,187, 210,201,192,
206,197,192, 208,205,196, 196,192,183, 204,193,175, 244,231,196, 251,232,189, 217,194,142, 194,173,126, 229,209,158, 242,223,167,
190,174,125, 93,74,44, 54,39,20, 73,60,41, 115,99,65, 169,147,98, 208,185,131, 218,199,141, 215,195,142, 205,181,133,
167,144,103, 148,126,85, 145,122,81, 139,116,74, 144,120,82, 130,107,73, 116,96,59, 129,107,68, 135,112,78, 138,112,79,
115,88,59, 84,63,42, 74,59,36, 76,63,44, 64,50,37, 52,40,26, 51,39,25, 72,61,43, 73,62,44, 62,48,37,
53,39,26, 55,42,26, 67,51,36, 69,52,32, 82,65,39, 96,79,53, 81,65,40, 67,50,32, 64,51,35, 62,50,38,
53,43,33, 53,40,34, 44,34,25, 44,34,25, 54,40,31, 53,39,30, 43,32,28, 43,33,31, 40,29,27, 39,29,27,
43,34,25, 49,40,35, 44,36,34, 29,25,24, 30,20,21, 33,23,24, 39,34,30, 52,43,34, 62,47,42, 71,55,42,
82,64,44, 78,58,47, 54,40,27, 43,32,30, 41,30,26, 44,34,24, 45,35,25, 44,36,25, 65,49,36, 86,61,41,
98,77,50, 94,72,48, 95,68,51, 95,67,55, 79,53,40, 69,42,25, 101,75,50, 139,114,83, 136,109,79, 157,128,98,
146,115,86, 171,140,112, 191,164,137, 153,132,103, 129,112,86, 135,119,103, 137,126,120, 141,132,125, 153,144,135, 107,98,91,
117,110,104, 105,101,92, 98,93,89, 98,95,88, 100,96,87, 103,97,85, 102,97,91, 103,97,99, 109,102,96, 110,102,99,
112,105,99, 118,109,102, 121,112,105, 121,114,106, 123,116,110, 120,115,111, 198,188,187, 198,188,187, 195,188,182, 197,190,180,
199,195,183, 199,193,177, 202,196,184, 207,198,191, 203,194,189, 205,194,190, 208,198,196, 213,202,198, 209,200,193, 209,207,195,
207,203,194, 217,206,188, 231,219,179, 241,220,167, 207,181,130, 152,127,86, 164,140,94, 184,162,115, 151,133,97, 96,79,59,
```

Затем, я создал в директории сэмпла файл `cat_image.h` и поместил в него получившиеся данные.

Теперь, нужно немного изменить исходный код сэмпла:

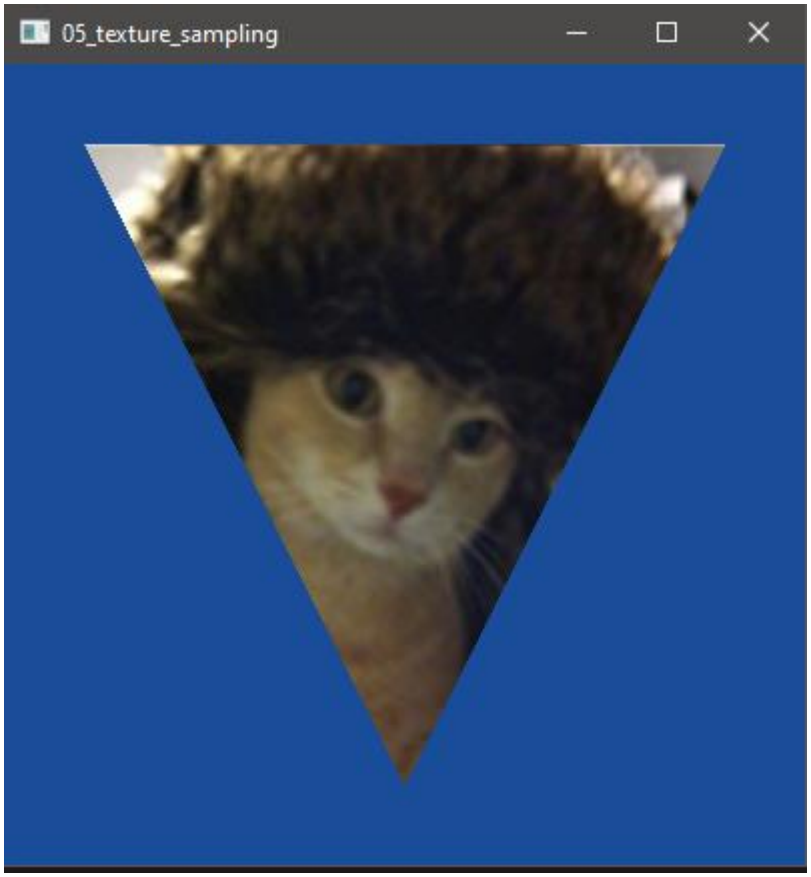
Находим данный блок:

```
static const unsigned char
myDemonTextureImage[3*(128*128+64*64+32*32+16*16+8*8+4*4+2*2+1*1)]
= {
/* RGB8 image data for a mipmapped 128x128 demon texture */
#include "demon_image.h"
};
```

Во-первых, меняем `demon_image.h` на `cat_image.h`.
Также видно, что массив myDemonTextureImage ожидает довольно большое количество данных, которых у нас нет. Теперь этот блок будет выглядеть так:

```
static const unsigned char
myCatTextureImage[3 * 128 * 128] = {
/* RGB8 image data for a mipmapped 128x128 cat texture */
#include "cat_image.h"
};
```

Это, в принципе, все изменения, которые нужно внести в исходный код, чтобы поменять текстуру.
Результат:



Далее, я должен показать passthrough-координаты, а также показать, как они пробегают все значения на экране. Для этого в проекте нужно будет изменить Вершинный и Фрагментный шейдеры.

В вершинном шейдере мы изменяем структуру Output, добавив ей новое поле:
`float4 rawPos    : TEXCOORD1;`
В котором будут сырые координаты. Естественно, их мы тоже должны будем передать дальше в OUT. После изменений шейдер `C3E3v_varying.cg` выглядит так:

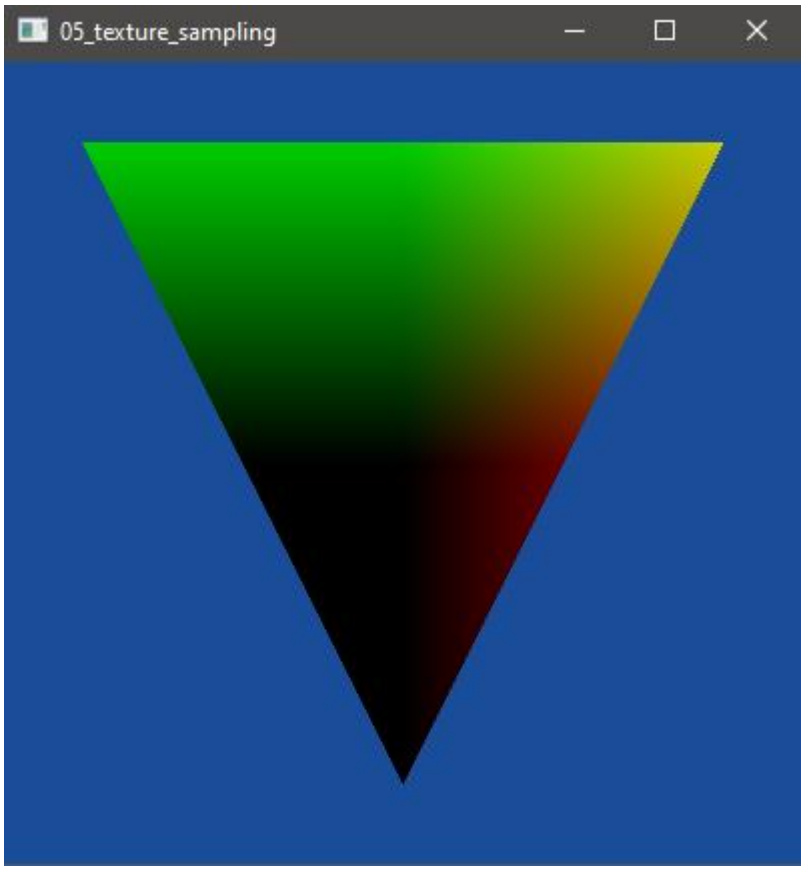
```
// This is C3E2v_varying from "The Cg Tutorial" (Addison-Wesley,
ISBN
// 0321194969) by Randima Fernando and Mark J. Kilgard.  See page
65.

struct C3E2v_Output {
  float4 position : POSITION;
  float3 color    : COLOR;
  float2 texCoord : TEXCOORD0;
```

```
  float2 rawPos   : TEXCOORD1;
};

C3E2v_Output C3E2v_varying(float2 position : POSITION,
                           float3 color    : COLOR,
                           float2 texCoord : TEXCOORD0)
{
  C3E2v_Output OUT;

  OUT.position = float4(position,0,1);
  OUT.color    = color;
  OUT.texCoord = texCoord;
  OUT.rawPos   = position;

  return OUT;
}
```

Теперь мы изменим Фрагментный шейдер (C3E3f_texture.cg):
Мы передаем в нее rawPos, который получили из вершинного шейдера, и выводим passthrough координаты:

```
// This is C3E3f_texture from "The Cg Tutorial" (Addison-Wesley,
ISBN
// 0321194969) by Randima Fernando and Mark J. Kilgard.  See page
67.

struct C3E3f_Output {
  float4 color : COLOR;
};

C3E3f_Output C3E3f_texture(float2 texCoord : TEXCOORD0,
                           float2 rawPos   : TEXCOORD1,
                           uniform sampler2D decal : TEX0)
{
  C3E3f_Output OUT;
  OUT.color = float4(rawPos.x, rawPos.y, 0.0, 1.0);
  return OUT;
}
```

Результат:

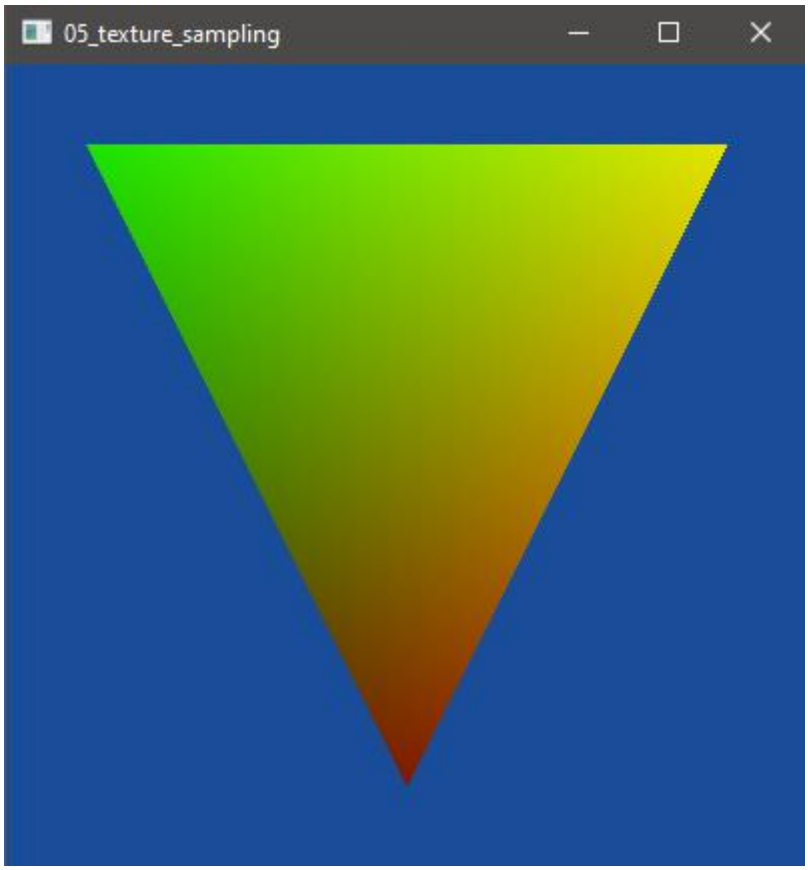


Теперь я должен реализовать формулу **0.5 * xy + 0.5,** чтобы нормализовать координаты и перевести диапазон из [-1, 1] в [1, 1]

Для этого нормализуем rawPos перед выводом, передав значение в переменные normalized:

```
{
  C3E3f_Output OUT;

  float2 normalized = rawPos * 0.5 + 0.5;

  OUT.color = float4(normalized.x, normalized.y, 0.0, 1.0);
  return OUT;
}
```

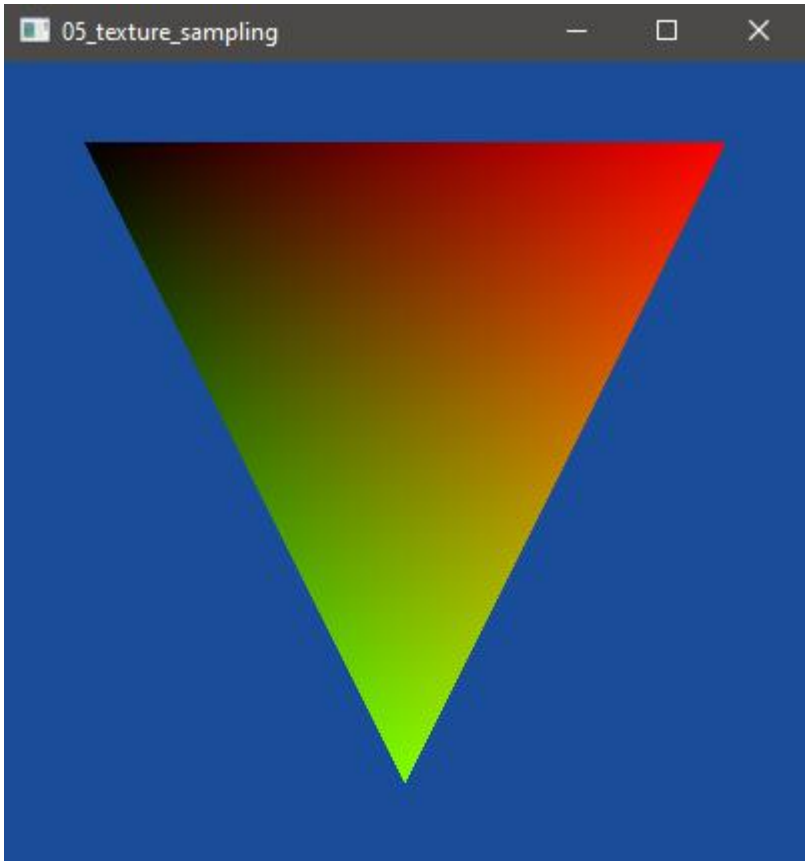
Результат:



Также мне нужно вывести UV-координаты. Для этого просто выводим texCoord, при этом они уже нормализованы:

```
{
  C3E3f_Output OUT;

  OUT.color = float4(texCoord.x, texCoord.y, 0.0, 1.0);
  return OUT;
}
```

Результат:



Итак, я вывел свою текстуру на треугольник, преобразовав изображение в byte-array, показал passthrough-координаты, нормализовал их, реализовав формулу 0.5 * xy + 0.5, и затем вывел UV-координаты.

Весь исходный код можно посмотреть тут:
https://github.com/seventwogth/graph-course