**OpenGL ES Shading Language 1.0**

**Mentor: Fikret Öztürk**

A black text on a white background

Description automatically generated

**Abdulkadir Bilge**

İçerikler

[3.4- Preprocessor 2](#_Toc142823164)

[Preprocessor Directives 2](#_Toc142823165)

[Predefined Macros 4](#_Toc142823166)

[4.1- Types 5](#_Toc142823167)

[Basic Types 5](#_Toc142823168)

[Structures: 13](#_Toc142823169)

[Arrays (Diziler): 13](#_Toc142823170)

[4.3- Storage Qualifiers 15](#_Toc142823171)

[4.4- Parameter Qualifiers 17](#_Toc142823172)

[4.5- Precision and Precision Qualifiers 17](#_Toc142823173)

[4.6- Invariant Qualifiers Examples 18](#_Toc142823174)

[4.7- Order of Qualification 19](#_Toc142823175)

[5.1- Operators 20](#_Toc142823176)

[5.4- Matrix Constructor Examples 21](#_Toc142823177)

[5.5- Vector Components 21](#_Toc142823178)

[5.6- Matrix Components 22](#_Toc142823179)

[5.7- Structure Operations 22](#_Toc142823180)

# 3.4- Preprocessor

## Preprocessor Directives

**#define:** Kod boyunca kullanılabilecek makroyu tanımlar.

**Örnek Kullanım:**

#define MAX\_LIGHTS 8

**#undef:** Daha önceden tanımlanmış makronun tanımını kaldırır.

**Örnek Kullanım:**

#undef MAX\_LIGHTS

**#error:** Derleme hatasını özel bir mesajla gösterir.

**#ifdef:** Tanımlı bir makro varsa. Koşullu derlemeyi aktif hale getirir.

**#ifndef:** Tanımlı bir makro yoksa. Koşullu derlemeyi inaktif hale getirir.

**#if #endif:** If endif bloğu oluşturur.

**Örnek Kullanım:**

#if defined(UNSUPPORTED\_GPU)

    #error This shader is not supported on your GPU.

#endif

**#ifdef**  **#else # endif:** Ifdef else endif bloğu oluşturur.

**Örnek Kullanım:**

#ifdef SHADOWS\_ENABLED

    float shadowIntensity = 0.5;

#else

    float shadowIntensity = 1.0;

#endif

**#ifndef #endif:** Ifndef endif bloğu oluşturur.

**Örnek Kullanım:**

#ifndef TEXTURE\_ENABLED

    #define DEFAULT\_COLOR vec3(1.0, 0.0, 0.0)

#endif

**#ifdef #elif #else #endif:** Ifdef elif else endif bloğu oluşturur.

**Örnek Kullanım:**

#ifdef SPECULAR\_ENABLED

    float shininess = 32.0;

#elif defined DIFFUSE\_ENABLED

    float shininess = 16.0;

#else

    float shininess = 8.0;

#endif

**#pragma:** Derleyiciye spesifik uygulamada bulunmak için ipucu verir.

**Örnek Kullanım:**

#pragma optimize(on)

#pragma debug(on)

**#extension:** Spesifik uzantıları aktif eder veya devre dışı bırakır.

**Örnek Kullanım:**

#extension GL\_EXT\_shader\_texture\_lod : enable

**#version:** Shader için GLSL versiyonu belirler.

**Örnek Kullanım:**

#version 330 core

**#line:** Hata mesajı veya hata ayıklama için bulunduğu satır numarasını değiştirir.

**Örnek Kullanım:**

for (int i = 0; i < 10; i++) {

// ...

#line 120

}

void main() {

// ...

}

## Predefined Macros

**\_\_LINE\_\_:** Derlenen kodun bulunduğu satırın numarasını verir.

**Örnek Kullanım:**

void main() {

int lineNumber = \_\_LINE\_\_;

}

**\_\_FILE\_\_:** Derlenen kodun kaynak dosyasının ismini verir.

**Örnek Kullanım:**

void main() {

const char\* fileName = \_\_FILE\_\_;

}

**\_\_VERSION\_\_:** Kullanılan GLSL derleyicisinin versiyonunun numarasını verir.

**Örnek Kullanım:**

void main() {

int glslVersion = \_\_VERSION\_\_;

}

**GL\_ES:** OpenGL ES için shaderın derlenmesini kontrol eder.

**Örnek Kullanım:**

void main() {

#ifdef GL\_ES

     gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);

#else

     gl\_FragColor = vec4(0.0, 0.0, 1.0, 1.0);

#endif

}

**GL\_FRAGMENT\_PRECISION\_HIGH:** Fragment shader hesaplamaları için yüksek hassasiyet gerektiğini belirtir.

**Örnek Kullanım:**

void main() {

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

     #pragma highp

#endif

gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);

}

# 4.1- Types

## Basic Types

**Void:**

Geri döngüsü olmayan fonksiyonu tanımlar. Yakarıda birçok örneği mevcut.

**Bool:**

Boolean (true-doğru ve false-yanlış) tipini belli eder.

**Int:**

Integer (tam sayı) tipini belli eder.

**Float:**

Float (1.0) tipindeki sayıları belli eder.

**Vec:**

Vec2 vec3 ve vec4 matematikte gördüğümüz vektörleri ifade eder. Vec2 2 boyutlu vektörü vec3 3 boyutlu vektörü ve vec4 4 boyutlu vektörü ifade eder. Bu vektörleri kullanarak farklı şeyler ifade edebiliriz. Bu vektörlerin parametreleri float tipindeki sayılarıdır.

**Vec2:**

Vec2 kullanarak 2 boyutlu vektörün yansıra konumu ve doku koordinatını ifade edebiliriz

Aşağıda nasıl ifade edilebileceği gösterilmektedir.

**Örnek Kullanım:**

vec2 position = vec2(1.0, 1.0);

vec2 texCoord = vec2(1.0, 1.0);

vec2 sum = position + texCoord;

vec2 scaled = position \* 0.5;

float length = length(position);

float x = position.x;

float y = position.y;

Matematikte gerçekleştirdiğimiz vektör operasyonlarını da gerçekleştirebiliriz ve bileşenlerine erişebiliriz. Örneğin, Vec2 sum komutu sum degişkeni adı altında 2 boyutlu vektör olarak tanımlıyor. Bu vektörün bileşenlerinde 2 boyutlu konum vektörünün ve doku koordinatlarının toplamıdır.

**Vec3:**

Vec3 3 boyutlu vektör olarak tanımlanır ve 3 boyutlu konumu tanımlayabilir. Ayrıca RGB cinsinden renkleri de tanımlayabiliriz ve vec2 ile yapılan tum matematiksel operasyonları gerçekleştirebilir.

**Örnek Kullanım:**

vec3 position = vec3(1.0, 2.0, -0.5);

vec3 color = vec3(0.8, 0.2, 0.5);

vec3 sum = position + color;

vec3 scaled = position \* 2.0;

float length = length(position);

float x = position.x;

float y = position.y;

float z = position.z;

**Vec4:**

Vec4 4 boyutlu vektör olarak tanımlanır ve 4 boyutlu konumu tanımlayabilir. Ayrıca RGB cinsinden renkleri de tanımlayabiliriz ve vec2-vec3 ile yapılan tüm matematiksel operasyonları gerçekleştirebilir.

**Örnek Kullanım:**

vec4 vector = vec4(1.0, 2.0, 0.5, 1.0);

vec4 color = vec4(0.8, 0.2, 0.5, 1.0);

vec4 sum = vector + color;

vec4 scaled = vector \* 0.5;

float dotProduct = dot(vector, color);

float x = vector.x;

float y = vector.y;

float z = vector.z;

float w = vector.w;

**Bvec:**

Bvec2, bvec3, ve bvec4 in yukarıda bahsedilen vektör komutlarından farkı parametrelerdir. Vec komutuyla elde ettiğimiz vektörlerin parametreleri float tipinde sayı iken bvec komutunun parametreleri boolean (true ya da false) olarak verilmek zorundadır.

**bvec2:**

**Örnek Kullanım:**

bvec2 condition = bvec2(true, false);

Parametrelerimiz sayı olmadığı için bvec2 için matematiksel operasyon yapılamaz.  Sadece vektör olarak tanımlanabilir.

**bvec3:**

**Örnek Kullanım:**

bvec3 condition = bvec3(true, false, true);

Parametrelerimiz sayı olmadığı için bvec3 için matematiksel operasyon yapılamaz. Sadece vektör olarak tanımlanabilir.

**Bvec4:**

**Örnek Kullanım:**

bvec4 condition = bvec4(true, true, false, false);

Parametrelerimiz sayı olmadığı için bvec4 için matematiksel operasyon yapılamaz. Sadece vektör olarak tanımlanabilir.

**Ivec:**

**Ivec2:**

Ivec2 ile vec2 birbirine çok benzemektedir. Tek farkı vec2 nin parametreleri float tipindeki sayılar iken ivec2 nin parametreleri integer tipindeki sayılardır.

**Örnek Kullanım:**

ivec2 position = ivec2(300, 450);

ivec2 velocity = ivec2(5, -2);

int x = position.x;

int y = position.y;

ivec2 texCoord = ivec2(20, 10);

ivec2 texSize = ivec2(512, 512);

ivec2 sum = position + velocity;

ivec2 scaled = position \* 2;

**Ivec3:**

Ivec3 ile vec3 birbirine çok benzemektedir. Tek farkı vec3 nin parametreleri float tipindeki sayılar iken ivec3 nin parametreleri integer tipindeki sayılardır.

**Örnek Kullanım:**

ivec3 position = ivec3(10, -5, 2);

int x = position.x;

int y = position.y;

int z = position.z;

ivec3 colorIndices = ivec3(2, 1, 0);

ivec3 sum = position + dimensions;

ivec3 scaled = position \* 2;

**Ivec4:**

Ivec4 ile vec4 birbirine çok benzemektedir. Tek farkı vec4 nin parametreleri float tipindeki sayılar iken ivec4 nin parametreleri integer tipindeki sayılardır.

**Örnek Kullanım:**

ivec4 color = ivec4(255, 128, 0, 255);

int r = color.r;

int g = color.g;

int b = color.b;

int a = color.a;

ivec4 position = ivec4(10, -5, 2, 1);

int w = position.w;

ivec4 dimensions = ivec4(800, 600, 200, 0);

ivec4 result = position + dimensions;

ivec4 scaled = position \* 2;

**Mat:**

**Mat2:**

Bu komut matematikte gördüğümüz matrisin bire bir aynisi. Bu fonksiyonun parametreleri de float tipindeki sayılardır ve tüm matrisler kare matristir. Parametreler aşağıda belirtildiği gibi ya da direkt yan yana yazılabilir. Daha rahat işlem yapılabilmesi için aşağıdaki gibi yazılmalı. Parametreler direkt virgülle ayrılır ve mat2 için ilk iki parametre ilk satiri diğer iki parametre son satiri oluşturur. Matematiksel işlemlerin tamamı yapılabilir.

**Örnek Kullanım:**

mat2 rotationMatrix = mat2(

0.866, -0.500,

0.500, 0.866

);

mat2 scalingMatrix = mat2(

2.0, 0.0,

0.0, 1.5

);

vec2 originalVector = vec2(1.0, 0.0);

vec2 transformedVector = rotationMatrix \* originalVector;

**Mat3:**

Bu komut matematikte gördüğümüz matrisin bire bir aynisi. Bu fonksiyonun parametreleri de float tipindeki sayılardır ve tüm matrisler kare matristir. Parametreler aşağıda belirtildiği gibi ya da direkt yan yana yazılabilir. Daha rahat işlem yapılabilmesi için aşağıdaki gibi yazılmalı. Parametreler direkt virgülle ayrılır ve mat3 için ilk üç parametre ilk satiri sonraki üç parametre ikinci satiri ve son üç parametre son satiri oluşturur. Matematiksel işlemlerin tamamı yapılabilir.

**Örnek Kullanım:**

mat3 transformationMatrix = mat3(

1.0, 0.0, 0.0,

0.0, 1.0, 0.0,

0.2, 0.3, 1.0

);

mat3 rotationMatrix = mat3(

0.866, -0.500, 0.0,

0.500, 0.866, 0.0,

0.0, 0.0, 1.0

);

vec3 originalVector = vec3(1.0, 0.0, 0.0);

vec3 transformedVector = transformationMatrix \* originalVector; //

mat3 normalMatrix = transpose(inverse(mat3(

0.866, -0.500, 0.0,

0.500, 0.866, 0.0,

0.0, 0.0, 1.0

)));

**Mat4:**

Bu komut matematikte gördüğümüz matrisin bire bir aynisi. Bu fonksiyonun parametreleri de float tipindeki sayılardır ve tüm matrisler kare matristir. Parametreler aşağıda belirtildiği gibi ya da direkt yan yana yazılabilir. Daha rahat işlem yapılabilmesi için aşağıdaki gibi yazılmalı. Parametreler direkt virgülle ayrılır ve mat4 için ilk dört parametre ilk satiri sonraki dört parametre ikinci satiri, sonraki dört parametre üçüncü satiri ve son dört parametre son satiri oluşturur. Matematiksel işlemlerin tamamı yapılabilir.

**Örnek Kullanım:**

mat4 projectionMatrix = mat4(

1.732, 0.0, 0.0, 0.0,

0.0, 1.732, 0.0, 0.0,

0.0, 0.0, -1.001, -1.0,

0.0, 0.0, -0.1001, 0.0

);

mat4 viewMatrix = mat4(

1.0, 0.0, 0.0, -2.0,

0.0, 1.0, 0.0, 3.0,

0.0, 0.0, 1.0, -5.0,

0.0, 0.0, 0.0, 1.0

);

mat4 modelMatrix = mat4(

1.0, 0.0, 0.0, 1.0,

0.0, 1.0, 0.0, -2.0,

0.0, 0.0, 1.0, 0.0,

0.0, 0.0, 0.0, 1.0

);

mat4 mvpMatrix = projectionMatrix \* viewMatrix \* modelMatrix;

vec4 originalVector = vec4(1.0, 0.0, 0.0, 1.0);

vec4 transformedVector = mvpMatrix \* originalVector;

**sampler2D:**

GLSL'de "sampler2D", bir 2 boyutlu doku örnekleyiciyi temsil etmek için kullanılan bir veri türüdür. OpenGL'deki dokulara genellikle shaderdaki örnekleyici değişkenler kullanılarak erişilir. `sampler2D` tipi, doku öğesi verilerini 2 boyutlu dokudan almanızı ve shader hesaplamalarınızda kullanmanızı sağlar. Genellikle doku eşleme gibi, yüzeylere uygulamak için dokulardaki renkleri örneklediğiniz görevler için kullanılır.

**Örnek Kullanım:**

**Fragment Shader:**

uniform sampler2D u\_texture;

uniform vec2 u\_resolution;

uniform float u\_time;

out vec4 FragColor;

void main(){

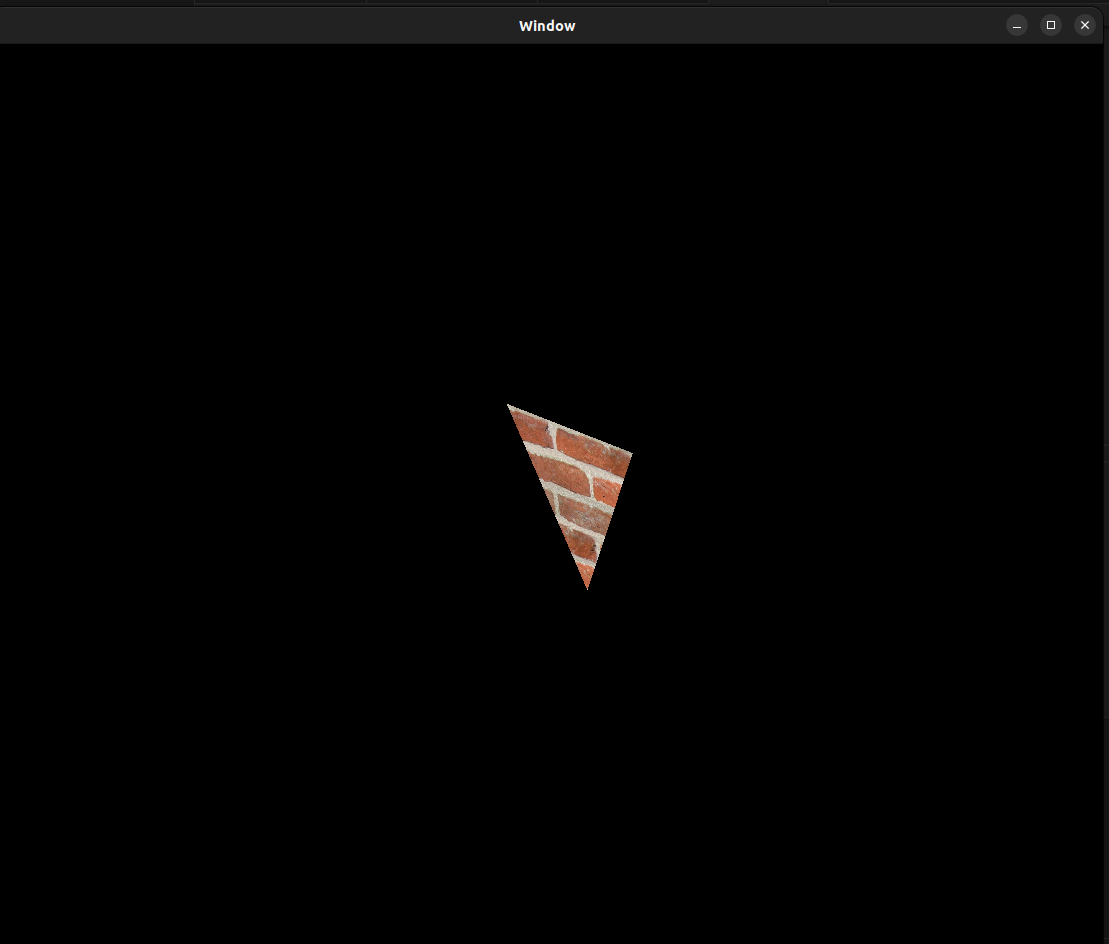
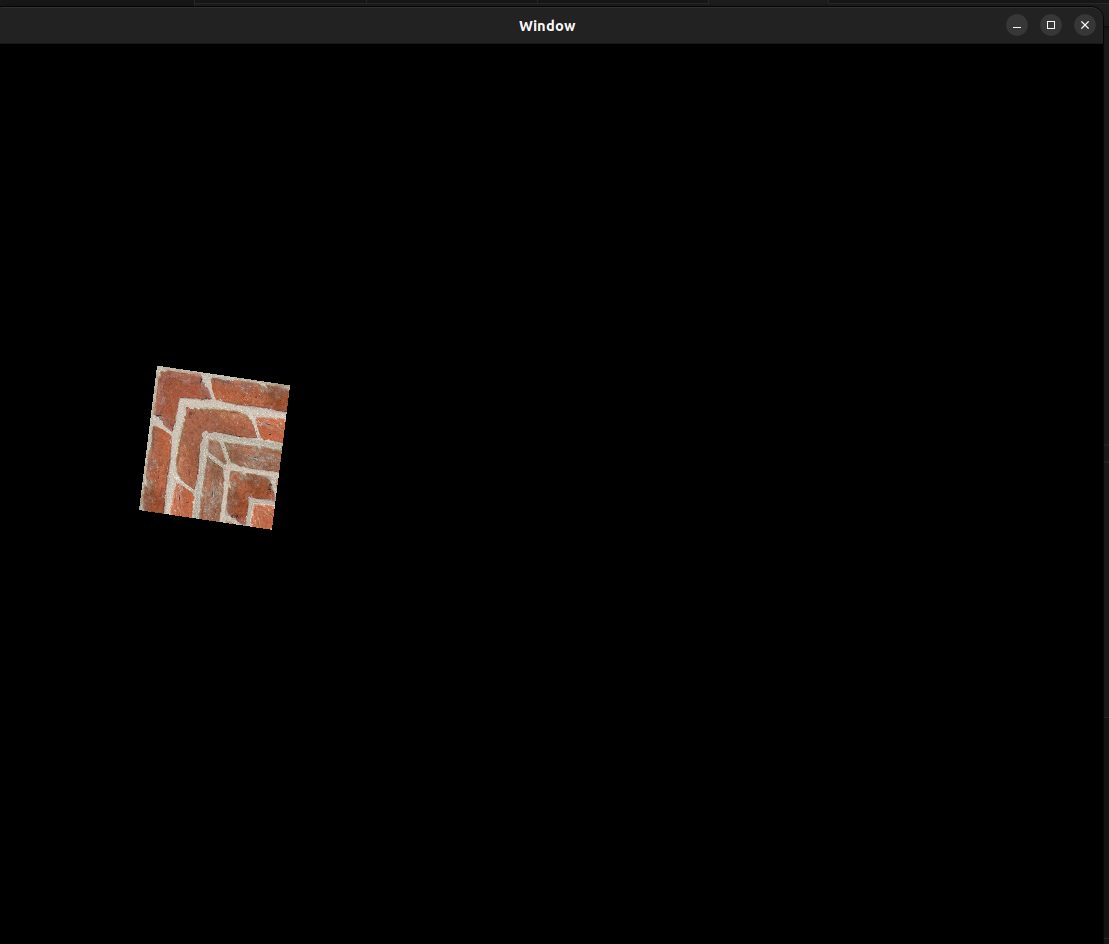
vec2 finalTexCoord = mix(TexCoord, FragTexCoord, 0.5);

vec4 textureColor = texture(u\_texture, finalTexCoord);

vec4 finalColor = textureColor;

FragColor = vec4(finalColor);

}



textureye renk eklemek icin maini assagidaki sekilde duzenleyin.

void main(){

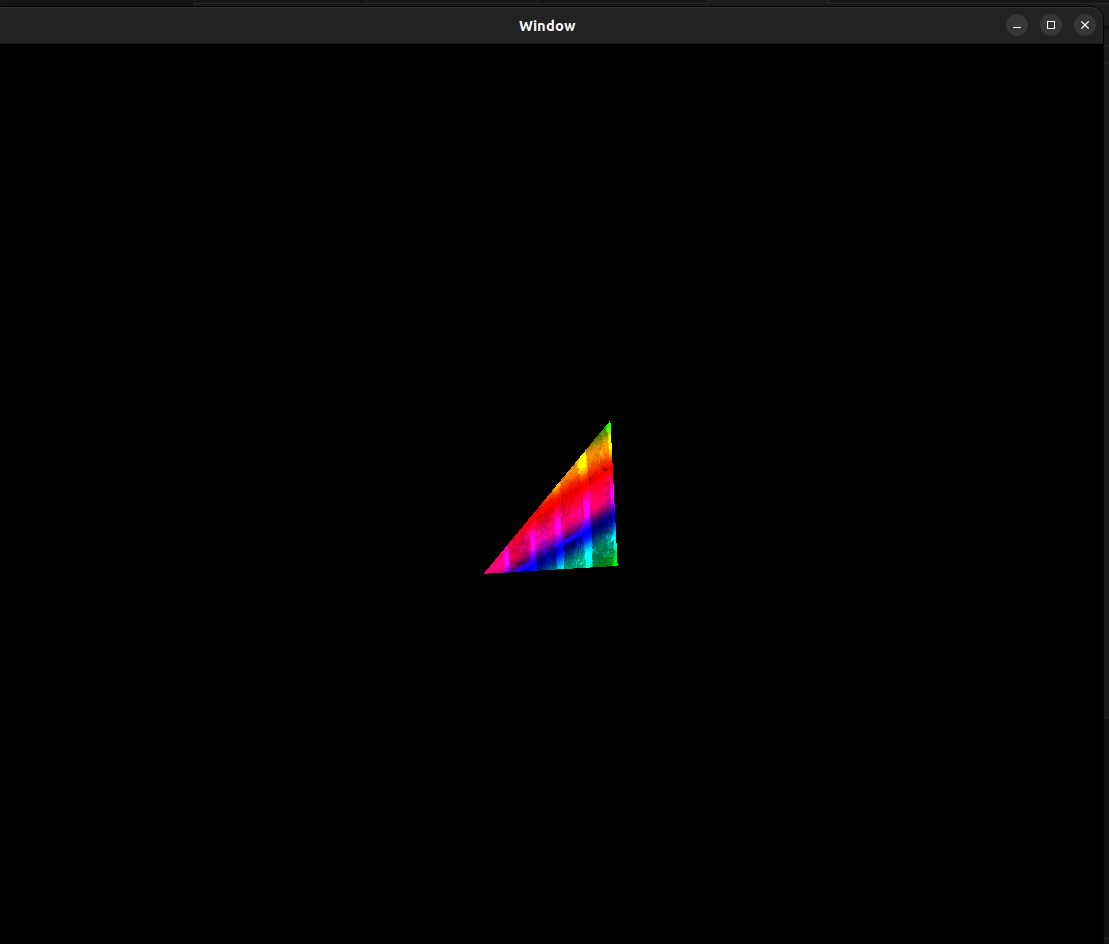
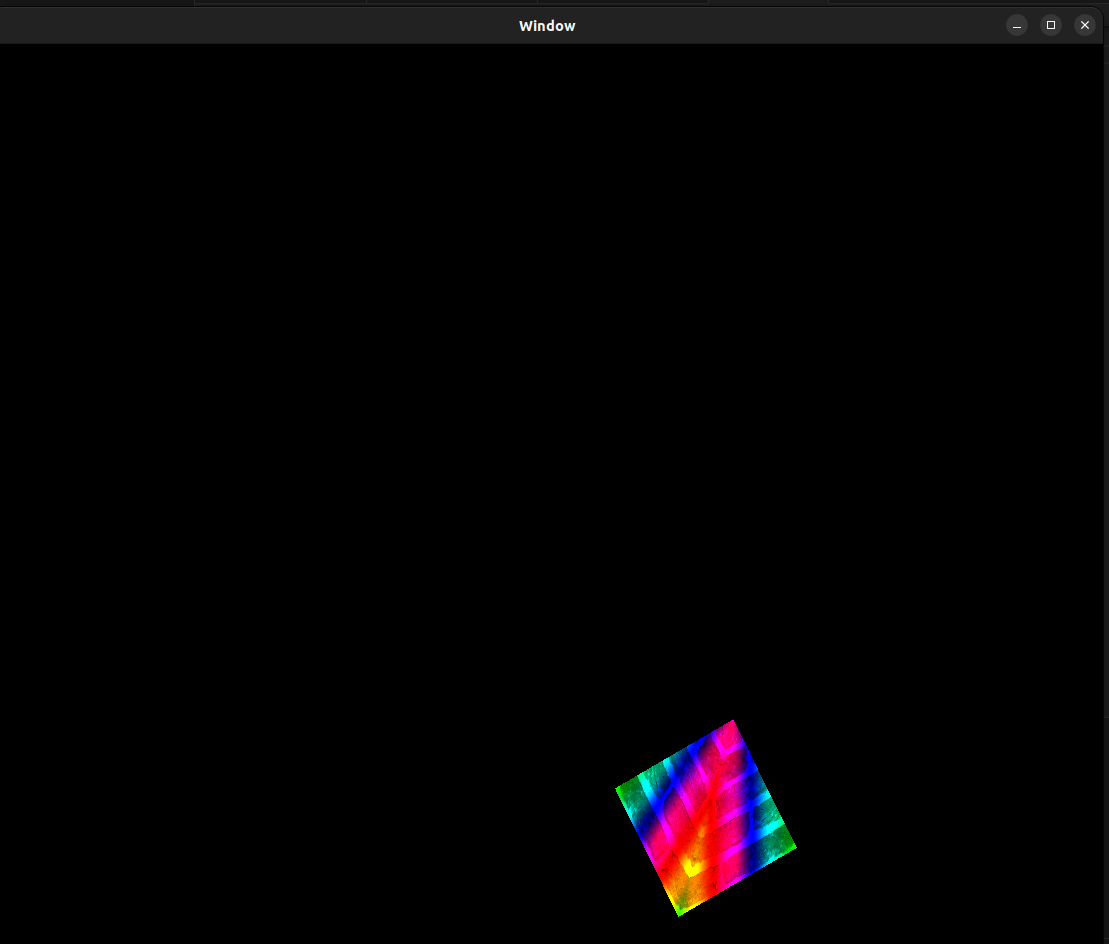
vec2 finalTexCoord = mix(TexCoord, FragTexCoord, 0.5);

vec4 textureColor = texture(u\_texture, finalTexCoord);

vec3 finalColor = textureColor.rgb \* rgbColor;

FragColor = vec4(finalColor, 1.0);

}



Vertex Shader ise kamera acisini ve konumu belirlemek icin kullanilir

uniform mat4 u\_modelMatrix;

uniform mat4 u\_viewMatrix;

uniform mat4 u\_projMatrix;

in vec3 inPosition;

out vec2 TexCoord;

in vec2 inTexCoord;

void main(){

gl\_Position = u\_projMatrix \* u\_viewMatrix \* u\_modelMatrix \* vec4(inPosition, 1.0);

TexCoord = inTexCoord;

}

**samplerCube:**

GLSL'de samplerCube, küp harita dokuları için bir örnekleyiciyi temsil etmek üzere kullanılan bir veri türüdür. Küp harita dokuları, yüzeylerdeki yansımaları veya kırılmaları simüle etmek için kullanılan, ortam haritalarını depolayan özel bir doku türüdür. Bir küp harita dokusu, her yüzün 3 boyutlu uzayda farklı bir yönü temsil ettiği, küp şeklinde düzenlenmiş altı 2 boyutlu görüntüden (yüz) oluşur.

**Örnek Kullanım:**

**fragment shader**

out vec4 FragColor;

in vec3 TexCoord;

uniform samplerCube u\_texture;

void main(){

vec3 direction = normalize(TexCoord-0.5);

FragColor=texture(u\_texture, direction);

}

**Vertex shader**

layout (location = 0) in vec3 aPos;

layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 u\_modelMatrix;

uniform mat4 u\_viewMatrix;

uniform mat4 u\_projMatrix;

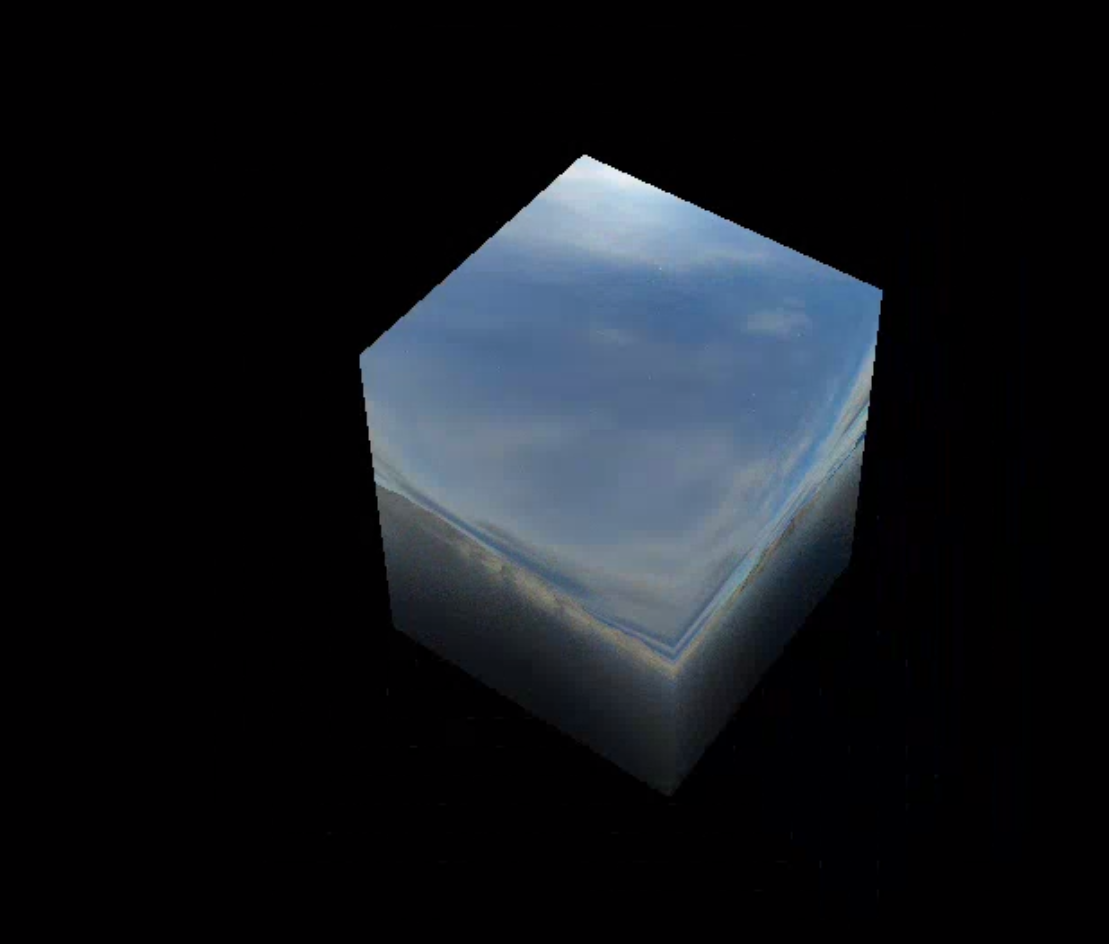
void main()

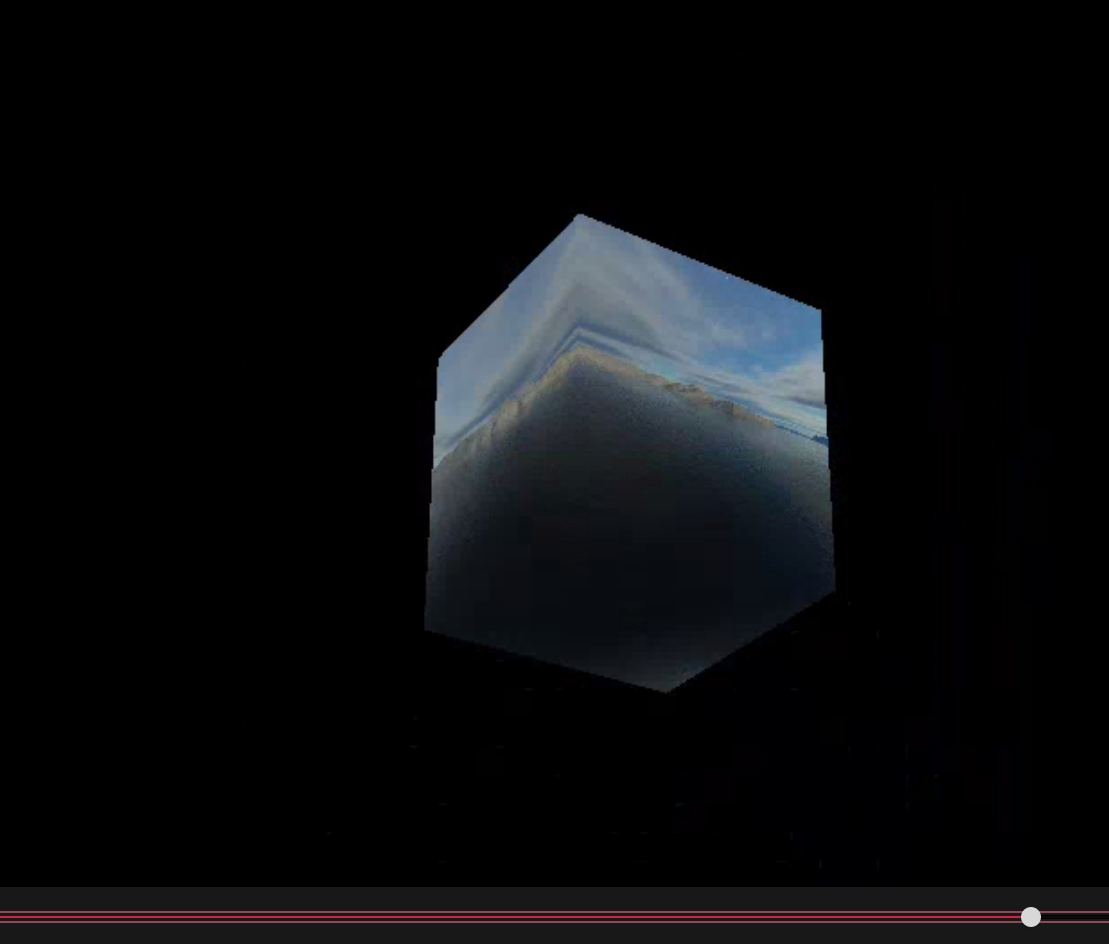
{

TexCoord = aTexCoord;

gl\_Position = u\_projMatrix \* u\_viewMatrix \* u\_modelMatrix \* vec4(aPos, 1.0);

}





Not: sampler2D ve samplerCube tiplerinin örneklerini daha iyi anlayabilmek için 4.3 ve 4.4’e göz atmalısınız.

## Structures:

Structures, verileri düzenlemenin ve kapsamanın bir yolunu sağlayarak kodunuzu daha düzenli ve okunabilir hale getirir. Yapılar, shaderlarin karmaşık verileri yönetmeyi ve değiştirmeyi kolaylaştırarak, birden çok özniteliği tutabilen özel veri türleri tanımlamanıza olanak tanır. Kullanımı aşağıda gösterilmiştir.

**Örnek Kullanım:**

struct Light {

vec3 position;

vec3 color;

float intensity;

};

Light myLight;

void main() {

myLight.position = vec3(1.0, 1.0, 1.0);

myLight.color = vec3(1.0, 0.0, 0.0);

myLight.intensity = 2.0;

vec3 lightDirection = normalize(myLight.position - vertexPosition);

float diffuseFactor = max(dot(normal, lightDirection), 0.0);

vec3 finalColor = myLight.color \* diffuseFactor \* myLight.intensity;

// ...

}

## Arrays (Diziler):

Diziler, değişken adından sonra dizinin boyutunu belirten köşeli parantezler [ ] kullanılarak bildirilir. Boyut sabit bir değer veya değişken olabilir.

**Örnek Kullanım:**

float myArray[5];

vec3 colorArray[3];

int dynamicArray[mySize];

**Dizi elementlerine erismek:**

**Örnek Kullanım:**

float value = myArray[2];

vec3 color = colorArray[0];

**Dizileri tanimlamak:**

**Örnek Kullanım:**

float myArray[5] = float[5](1.0, 2.0, 3.0, 4.0, 5.0);

vec3 colorArray[3] = vec3[3](

vec3(1.0, 0.0, 0.0),

vec3(0.0, 1.0, 0.0),

vec3(0.0, 0.0, 1.0)

);

**Örnek Kullanım:**

attribute vec3 positionArray[3];

attribute vec3 colorArray[3];

varying vec3 fragmentColor;

void main() {

fragmentColor = colorArray[gl\_VertexID];

gl\_Position = vec4(positionArray[gl\_VertexID], 1.0);

}

Dizilerin elemanlarına [ ] kullanarak erişebiliriz ve işlem yapabiliriz.

**Örnek Kullanım:**

diffuseColor += lightIntensity[3] \* NdotL

# 4.3- Storage Qualifiers

**Const:**

Bu niteleyici bir sabit tanımlamakta kullanılır ve tanımlanan bu sabit tanımlandıktan sonra değiştirilemez. Kodun optimizasyonuna ve okunabilirliğine katkı sağlamaktadır.

**Örnek Kullanım:**

const float PI = 3.14159;

const vec3 backgroundColor = vec3(0.0, 0.0, 0.0);

void calculateDiffuse(const vec3 lightDir, const vec3 normal) {

}

**Attribute:**

Sadece vertex shrader da kullanılır. Bu değişkenler, konum, normaller, doku koordinatları, renkler ve diğer vertexe özgü veriler gibi köşe başına öznitelikleri temsil eder. Niteliklerde depolanan veriler, her vertex için benzersizdir ve vertex shraderda işlemek için kullanılır.

**Örnek Kullanım:**

attribute vec3 position;

attribute vec2 texCoord;

varying vec2 fragTexCoord;

void main() {

gl\_Position = vec4(position, 1.0);

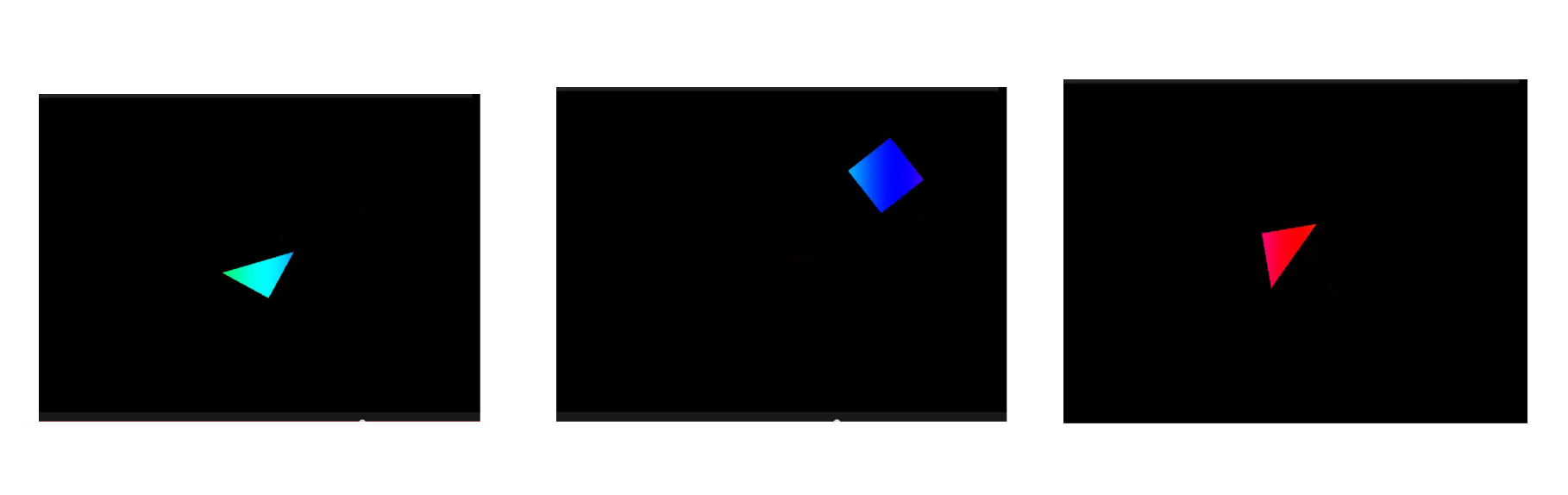
fragTexCoord = texCoord;

}

**Uniform**

Grafik kartının mimarisinden ötürü uniformu kullanmamız gereken durumlar vardır. Uniform kullanıldığı zaman grafik kartına giden veriler sadece okunabilir olur, değiştirilemez.

**Örnek Kullanım:**

****

Bu nesneler düzgün niteleyici tarafından yaratılmıştır. Dönüyorlar ve renkleri değişiyor. Bu döndürme ve renk değişimi kullanımın ne kadar düzgün olduğunu gösterir. Tek tip değişkenimiz zaman ve zaman, tüm gölgelendiriciler ve diğer dosyalar için aynıdır, kodu çalıştırdıktan sonraki süre aynıdır. Tek biçimli araçlar, gölgelendiriciler ve diğer kodlar arasında değişmediğinden, bu, tekdüzeliğin nasıl çalıştığını gösterir. Unifrom için kod kısmını aşağıda bulabilirsiniz:

glUniform1f(glGetUniformLocation(shaderUtil.GetProgramId(), "u\_time"), time); --> zaman verilerini gölgelendiricilere iletmek için.

glm::vec3 cameraUp = glm::vec3(sin(time), cos(time), 0.0f); --> nesnelerin dönmesini sağlamak

float waveFactor = sin(st.x \* 10.0 + u\_time) \* 0.1; --> dalgalı bir renk efekti vermek için

**Varying:**

Varying komutunu hem vertex shrader da hem de fragment shraderda kullanabildiğimiz bir komuttur ve bu iki shraderda ayni degişkeni kullanır.

**Örnek Kullanım:**

attribute vec3 position;

attribute vec2 texCoord;

varying vec2 fragTexCoord;

void main() {

gl\_Position = vec4(position, 1.0);

fragTexCoord = texCoord;

}

varying vec2 fragTexCoord;

uniform sampler2D textureSampler;

void main() {

vec4 texColor = texture2D(textureSampler, fragTexCoord);

gl\_FragColor = texColor;

}

Görüldüğü üzere fragTexCoord hem fragment shaderda hem de vertex shraderda kullanılmış ve fragTexCoord ikisinde de kolaylıkla kullanılabilir.

# 4.4- Parameter Qualifiers

**In and Out:**

In komutu girdi olarak düşünülebilir. Örneğin vertex shraderda out olan bir degişken fragment shraderda in olarak tanımlanıp kullanılabilir. Ayni şekilde fragmentte out olan bir şey vertexte in kullanılarak kullanılabilir. Shraderlar arasında veri geçişini mümkün kılar. Kısaca input output olarak kullanılır. Örnek kodda fragColor vektörü gibi.

**Örnek Kullanım:**

in vec3 position;

out vec3 fragColor;

void main() {

fragColor = position \* 0.5 + 0.5;

// ...

}

in vec3 fragColor;

void main() {

gl\_FragColor = vec4(fragColor, 1.0);

// ...

}

**Inout:**

Inout ise yukarıda bahsedilen in ve outun birleşmiş halidir. Inout sayesinde herhangi bir degişken hem in hem de out olacak şekilde kullanılabilir.

**Örnek Kullanım:**

inout vec3 position;

void modifyPosition() {

position.x = position.x \* 2.0;

}

void main() {

modifyPosition();

// ...

}

# 4.5- Precision and Precision Qualifiers

**Highp:**

Bu niteleyici, floating-point hesaplamaları için yüksek hassasiyet ister. Vertex dönüşümleri, aydınlatma hesaplamaları ve karmaşık matematiksel işlemler gibi yüksek doğruluk gerektiren hesaplamalar için uygundur.

**Mediump:**

Bu niteleyici, floating-point hesaplamaları için orta düzeyde kesinlik ister. Hassasiyet ve performans arasında bir denge sunar. Genellikle doku koordinatları ve ara hesaplamalar için kullanılır.

**Lowp:**

Bu niteleyici, floating-point hesaplamaları için düşük kesinlik ister. Renk hesaplamaları gibi kesinliğin kritik olmadığı durumlarda kullanılır ve performans için optimize edilmiştir.

**Örnek Kullanım:**

uniform highp float radiusHighp;

uniform mediump float radiusMediump;

uniform lowp float radiusLowp;

void main() {

highp float areaHighp = 3.141592653589793238 \* radiusHighp \* radiusHighp;

mediump float areaMediump = 3.141592653589793238 \* radiusMediump \* radiusMediump;

lowp float areaLowp = 3.141592653589793238 \* radiusLowp \* radiusLowp;

gl\_FragColor = vec4(areaHighp, areaMediump, areaLowp, 1.0);

}

Ciktilardaki degisik renkler hesaplama esnasinda olan hassasiyeti gosterir.

# 4.6- Invariant Qualifiers Examples

**#pragma STDGL invariant(all):**

#pragma yönergesi, GLSL derleyicisine göre veya shader işlem hattına özel yönergeler sağlamak için kullanılır. STDGL, "Standard Derivatives and Interpolation Functions” anlamına gelen bir GLSL uzantısıdır. Bir dizi standart davranış ve optimizasyon sağlar. invariant(all), shaderdaki tüm değişen değişkenler için değişmezliği zorunlu kılan bir talimattır. Değişmezlik, değişen değişkenlerin bir ilkelin tüm köşelerinde tutarlı değerlere sahip olmasını sağlar; bu, belirli iyileştirmeler için veya tutarlı shader sağlamak istediğinizde yararlı olabilir.

**invariant gl\_Position:**

gl\_Position'a uygulanan değişmez niteleyici, klip uzayındaki vertex konumu olan gl\_Position değerinin bir ilkelin tüm vertexlerde tutarlı olması gerektiğini belirtir. Bu, tutarlı vertex konumlarına bağlı dönüşümleri veya diğer işlemleri optimize etmek için yararlı olabilir.

**invariant varying mediump vec3 Color:**

Renk gibi değişen bir değişkene uygulanan değişmez niteleyici, bu değişkenin ilkel boyunca enterpolasyonu değerlerinin değişmez olması gerektiğini belirtir. Bu durumda, Color değişkeni için mediump kesinliği belirtilir. Bu niteleyici, değişen değişkenler için tutarlı enterpolasyon davranışının sağlanmasına yardımcı olur; bu, bir ilkel genelinde tutarlı shader veya efektler elde etmek için önemli olabilir.

**Örnek Kullanım:**

#pragma STDGL invariant(all)

invariant gl\_Position;

invariant varying mediump vec3 Color;

uniform mat4 modelViewProjection;

in vec3 inPosition;

in vec3 inColor;

out mediump vec3 Color;

void main() {

gl\_Position = modelViewProjection \* vec4(inPosition, 1.0);

Color = inColor;

}

# 4.7- Order of Qualification

Bu kısımda niteleyiciler birlikte kullanıldığı zaman hangisinin önceliği daha fazla onu incelemekteyiz.

Sırasıyla:

Invariant,

storage,

Precision

Ve aşağıdaki gibi:

storage,

parameter,

precision

Bu sıralamalar kesin olarak böyledir ve değiştirilemez.

# 5.1- Operators

|  |  |  |
| --- | --- | --- |
|  | Operatör | Tanım |
| 1. | () | Parantezle gruplama |
| 2. | []  ()  .  ++ -- | Dizi alt simgesi  Fonksiyon çağırma ve structure oluşturma  Alan veya metod seçme  Arttırma ve azaltma |
| 3. | ++ --  +- / | Arttırma ve azaltma  Tekli |
| 4. | \* / | Çarpım bölüm |
| 5. | + - | Toplama çıkarma |
| 7. | < > <= >= | İlişkiler |
| 8. | == != | Eşitlik |
| 12. | && | Mantıksal ve |
| 13. | ^^ | Mantıksal özel veya |
| 14. | || | Mantıksal dahil veya |
| 15. | ?: | Seçim |
| 16. | =  += -=  \*= /= | Aritmetik atama |
| 17. | , | Seriler |

# 5.4- Matrix Constructor Examples

**Örnek Kullanım:**

mat2(float)

mat2(vec2, vec2);

mat2(float, float,

float, float);

Bu örnekler mat2 kullanarak nasıl farklı şekillerde matris elde edebileceğimizi göstermektedir.

**Structure Constructor Example:**

**Örnek Kullanım:**

struct light {float intensity; vec3 pos; };

light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));

Yukarıdaki örnek structurenin kullanımını gösteren bir örnektir. Light isimli bir structure tanımlanmış ve bu structurenin ilk parametresi float ve diğeri 3 boyutlu vektör olarak tanımlanmış. Alt satırda ise lighVar isimli bir degişken bu structure içinde tanımlanmış. Parametrelerin ayni formatta olduğu görülebilir.

# 5.5- Vector Components

Vektör bileşenleri vektör boyutuna göre elemanlarına erişebilmek için kullanılan basit bir yoldur.

**Örnek Kullanım:**

{x,y,z,w} vektörün elemanlarına erişmek için .x .y .z veya .w  komutları kullanılır

Örneğin position={1.0,2.0,3.0,4.0} için positiyon.y 2.0 i verir.

{r, g, b, a} bileşenlerine yukarıda bahsedilen gibi erişebiliriz. Bu renklere erişmek için kullanılan versiyonudur.

{s, t, p, q} bileşenleri yukarıda bahsedilen gibi erişebiliriz. Bu doku koordinatlarına erişmek için kullanılan versiyonundur.

# 5.6- Matrix Components

mat4 m;  m matrisi ifade eder.

m[1] = vec4(2.0);  ikinci sütunu 2.0 olarak ayarlar.

m[0][0] = 1.0; sol üstteki elemanı 1.0 a eşitler

m[2][3] = 2.0;  3. Sütunun 4. Elemanını 2.0 a eşitler

m = f \* m;  sabit ve matris çarpımı

v = f \* v;  sabit ve vektör çarpımı

v = v \* v; vektör ve vektör çarpımı

m = m +/- m;  matrixler arası ekleme ve çıkarma

m = m \* m;  doğrusal cebir çarpımı

m = v \* m;  satır vektörü ve matris çarpımı

m = m \* v;  matrix ve sütun vektörü çarpımı

f = dot(v, v);  vektör nokta çarpımı

v = cross(v, v);  vektör çapraz çarpımı

m = matrixCompMult(m, m);  eleman çarpımı

# 5.7- Structure Operations

.  Alan seçimi

== !=  eşitlik

=  Görevlendirme

# Built-In Functions

Asagidaki tum fonksiyonlar direkt olarak kullanilarak elde edilmistir. Bir kismi icin detayli aciklama bulunmaktadir kalan fonksiyonlarda ayni sekilde elde edilmistir. Goruntunun hemen ustunde bulunan fonksiyonlar oldugu gibi kullanilmistir.

Ornegin float y = amplitude \* exp(x); fonksiyonu exp fonksiyoununu anlatmak icin kullanilmistir. Diger fonksiyonlar tanimlanirken degisen tek sey exp kismidir.

T radians (T degrees) degrees to radians :

Derece cinsinden olan açıyı radayana çevirir.

T degrees (T radians) radians to degrees:

Radyan cinsinden olan açıyı derecee çevirir.

T sin (T angle) sine:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Örneğin figur de sin kullanılarak çizilmiş bir sinus fonksiyonu var. Bu matematikten bildiğimiz sinüsün birebir aynısı. Bunu fragment shaderda bazı amaçlar için kullanabiliriz. Örneğin düz bir renk yerine değişen renkler yapabiliriz. Shaderlarda RGB kullanılır ve 1.0 0.0 0.0 kırmızı rengini 0.0 0.0 0.0 ise siyahı ifade eder Burada ilk değer yerine sinus koyduğumuz zaman kırmıız ve siyah arasında değişen bir renk elde ederiz. Figur de görüldüğü gibi

Kırmızı olan yerler sinus fonksiyonunun 1 değerinee yaklaştığı ve 1 değerini aldığı yerlerdir. 0 değerine yaklaştığında ve negative değerler aldığında ise siyah renk gözükmektedir. Frekansını arttırarak daha fazla geçiş görebilir, aşağıda gösterildiği gibi.

A screenshot of a computer

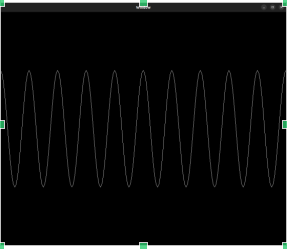
Description automatically generatedA screenshot of a computer

Description automatically generated

Dalga animasyonu gibi animasyonlarda yapılabilir.

T cos (T angle) cosine

Cosinuste sinle birebir aynı. Tek farkı aynı zamanda farklı değerler almaları. Figurlerde görüldüğü gibi



Figurlerde görüldüğü gibi birinde kırmızı renkle başlıyor cosinus çünkü 0 dayken 1 değerini alıyor sinus ise siyahla başlıyordu. Figurleri karşılaştırarak kolayca farkı görebiliriz.

T tan (T angle) tangent

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Tanjantın değerleri -sonsuz ile + sonsuz arasında olduğu için renk geçişleri çok daha net olur. Yine sinus ve cosinusteki gibi bir çıktı elde ederiz

Asagidaki fonksiyonlar sadece matematiksel islemler icin kullanilabilir

T asin(T x) arc sine

T acos(T x) arc cosine

T atan(T y, T x)

T atan(T y\_over\_x) arc tangent

T pow(T x, T y)

Figure shows .

A screen shot of a computer

Description automatically generatedA red and black rectangles

Description automatically generated

Bunun görüntüsü 0 ın yakınlarında siyah, 1 ve sonrasında kırmızı olacak şeilde olur aşağıdaki figurde görüldüğü gibi.

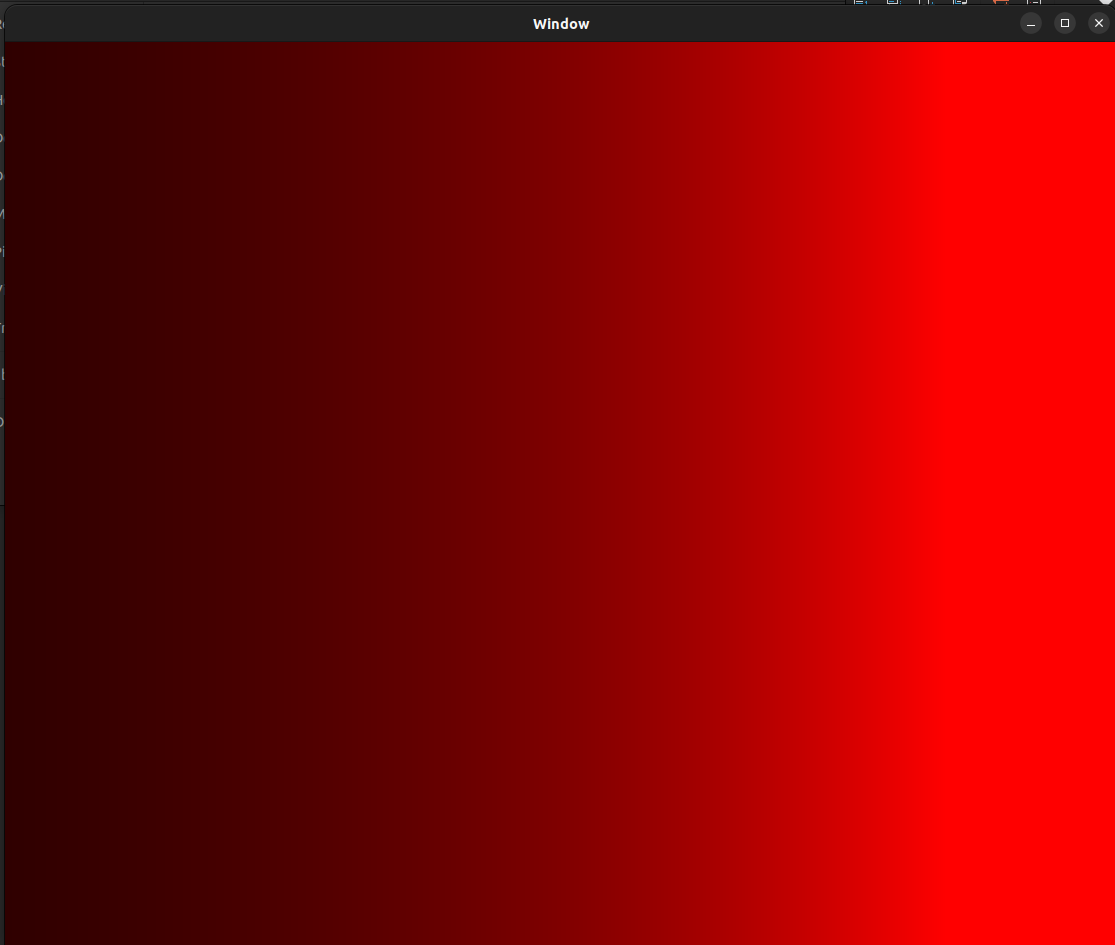
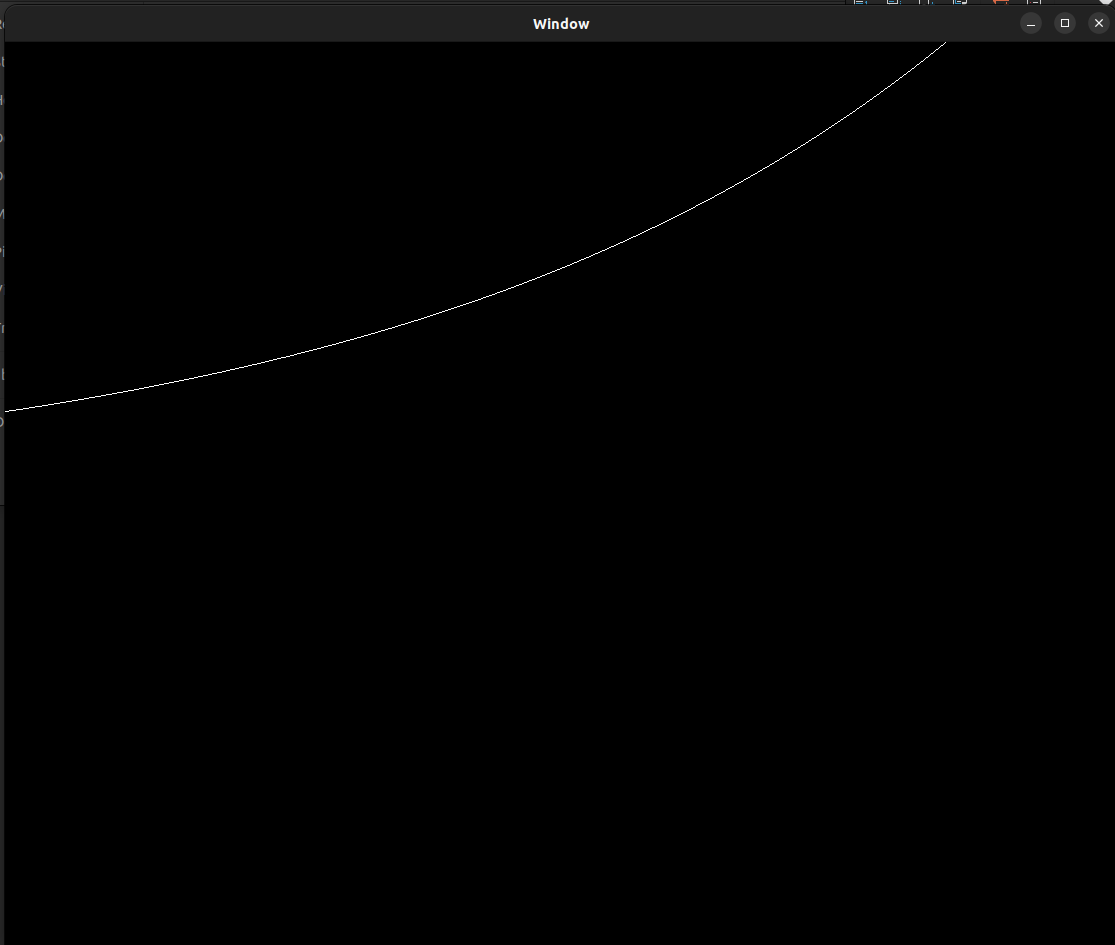
içinde benzer durum söz konusu.   
A screen shot of a computer

Description automatically generatedA screenshot of a computer

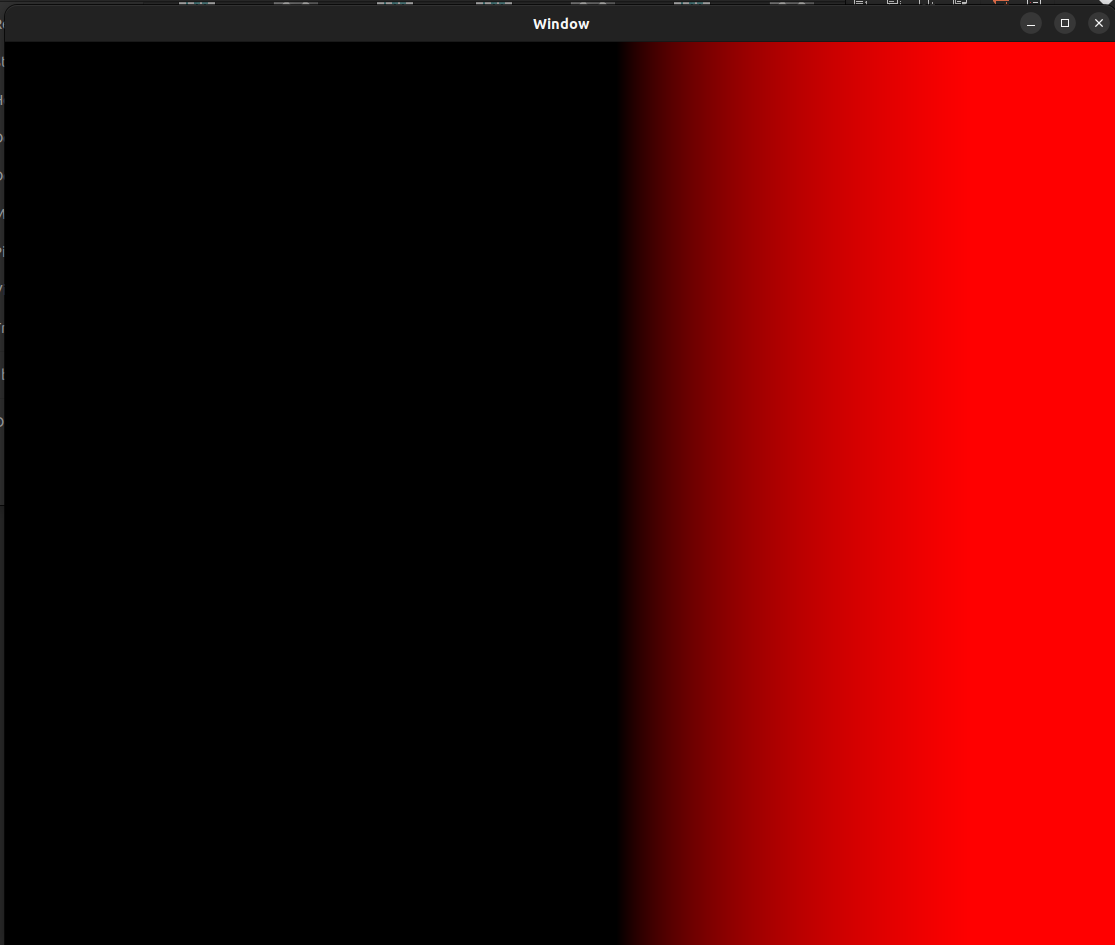
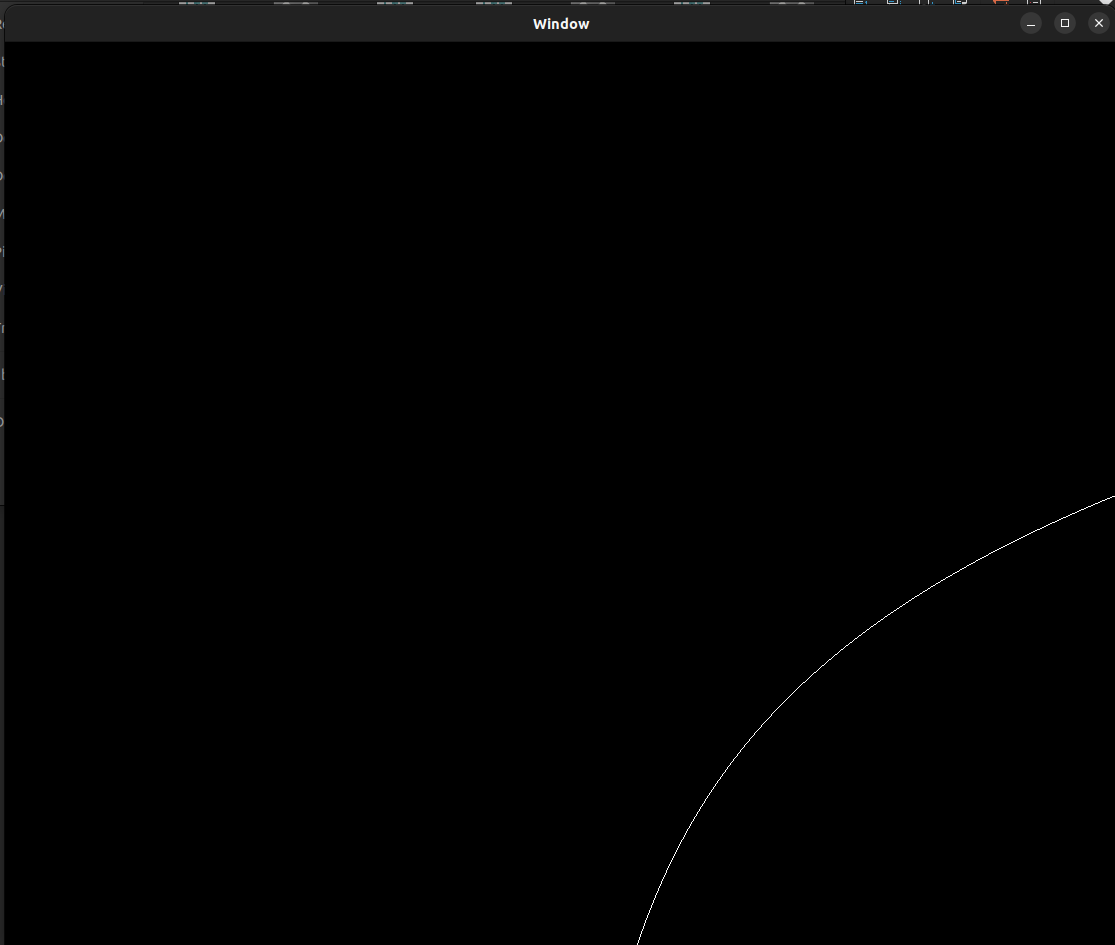
Description automatically generated

Bir yarısı pozitif diğer yarısı negatif olduğu için görüntünün yarısı siyah yarısı kırmızı olmalı figur gözüktüğü gibi.

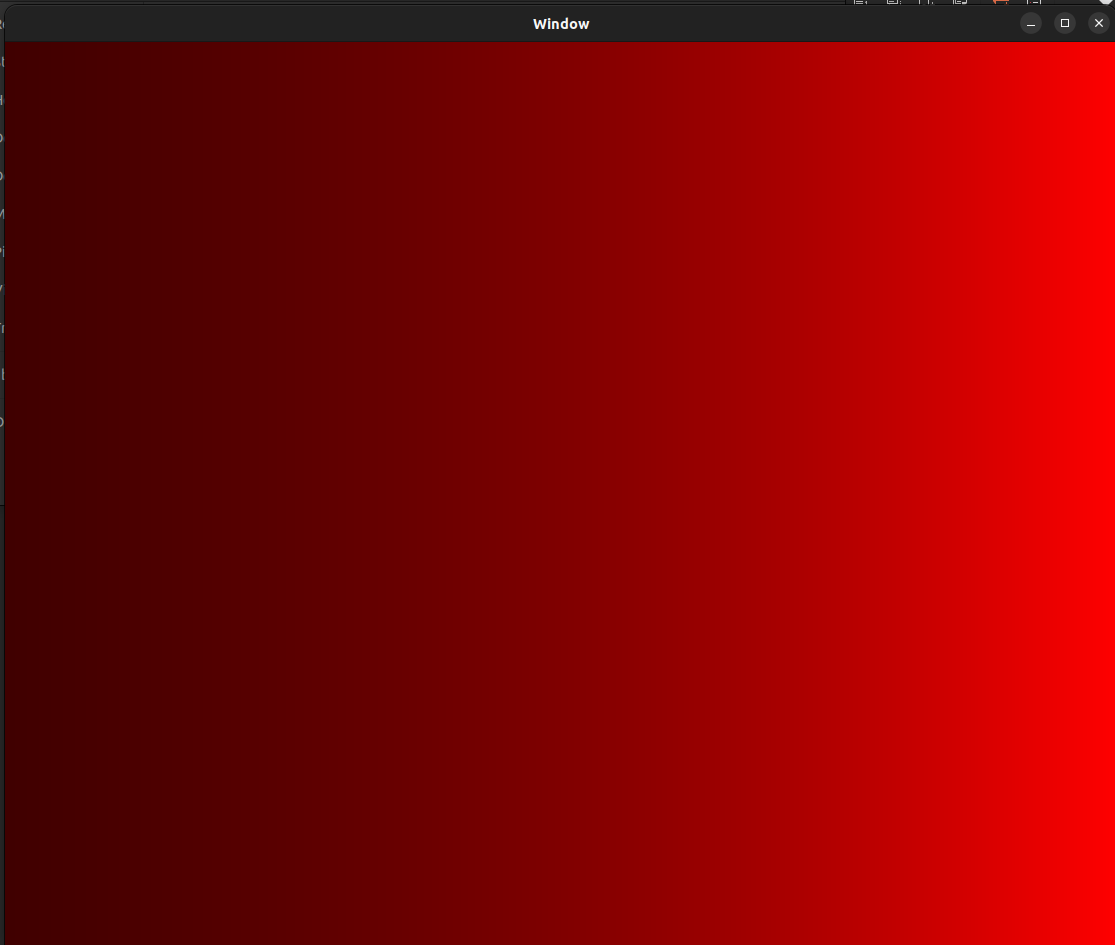
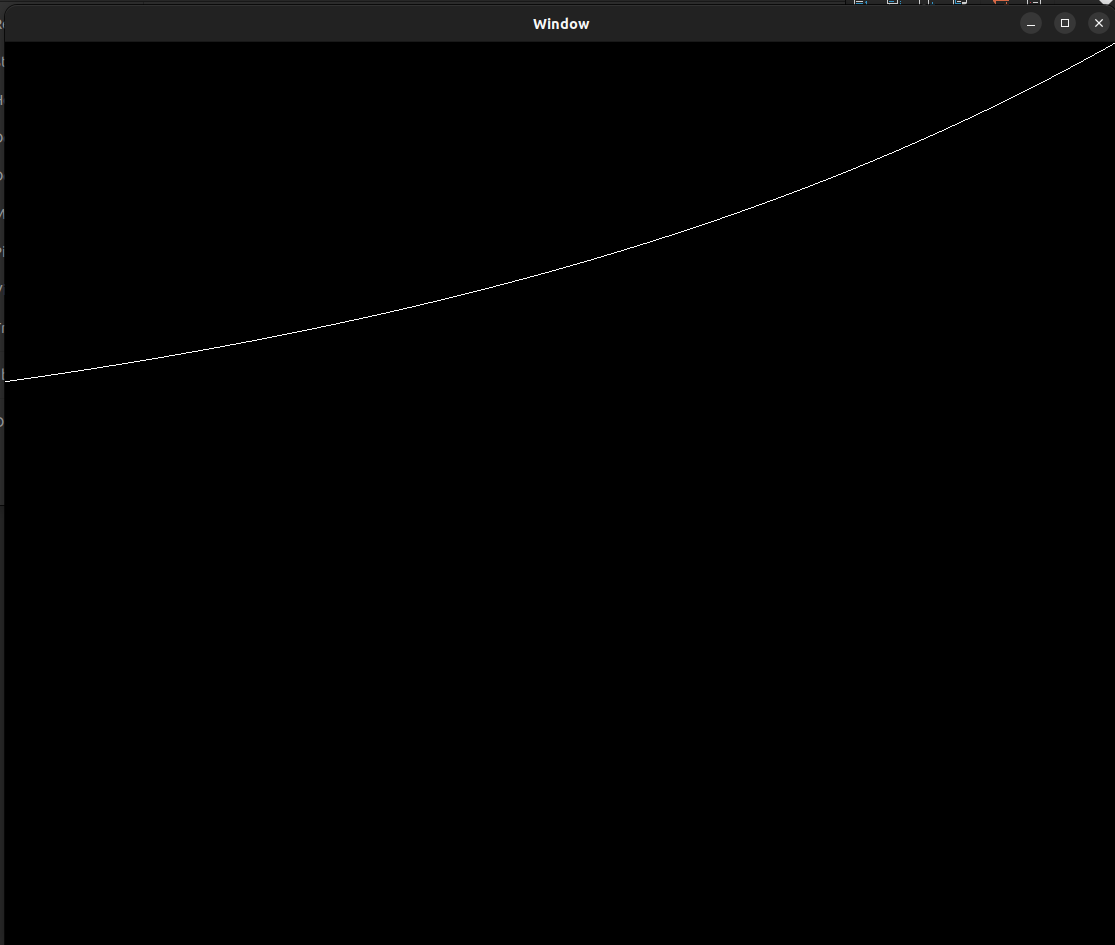
T exp(T x) ex

Exp fonksiyonun en kucuk degeri 1 oldugu icin siyah goruntu elde edemiyoruz.

T log(T x) ln

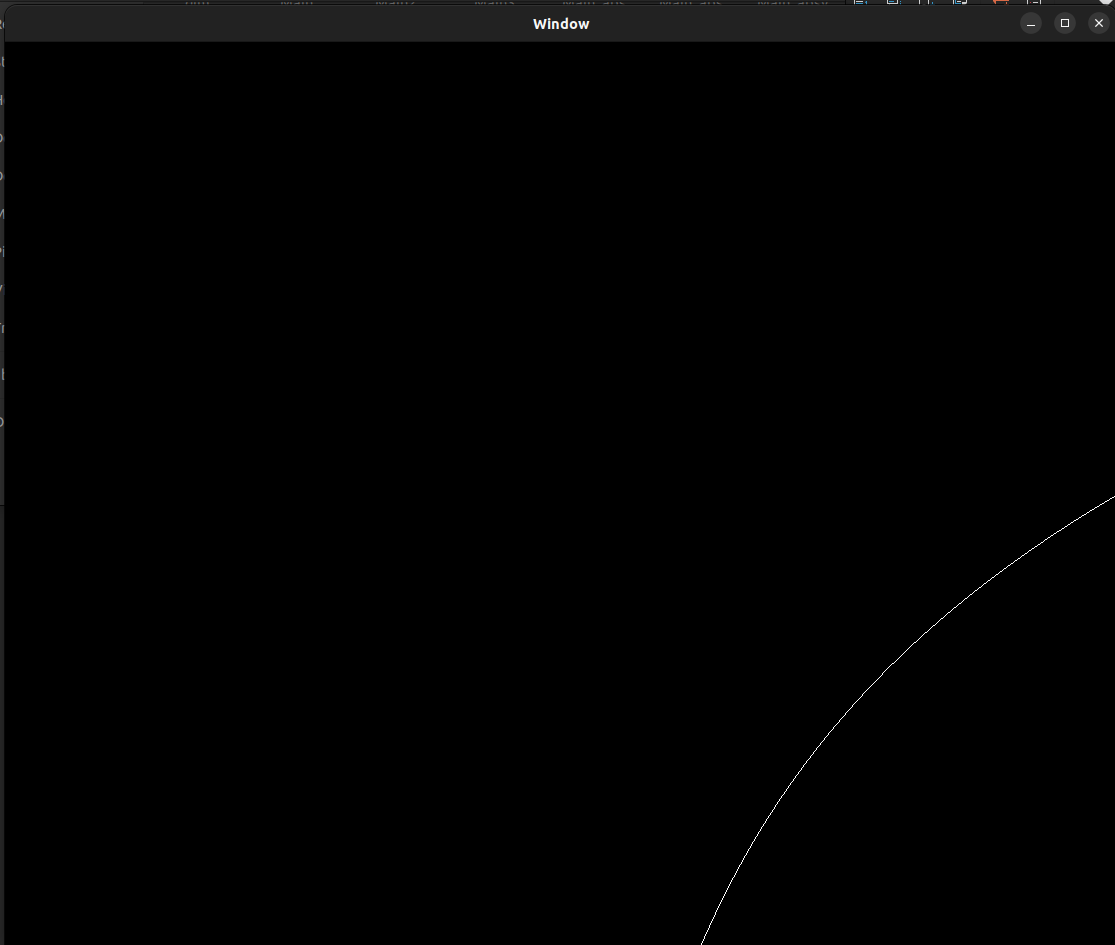


T exp2(T x)

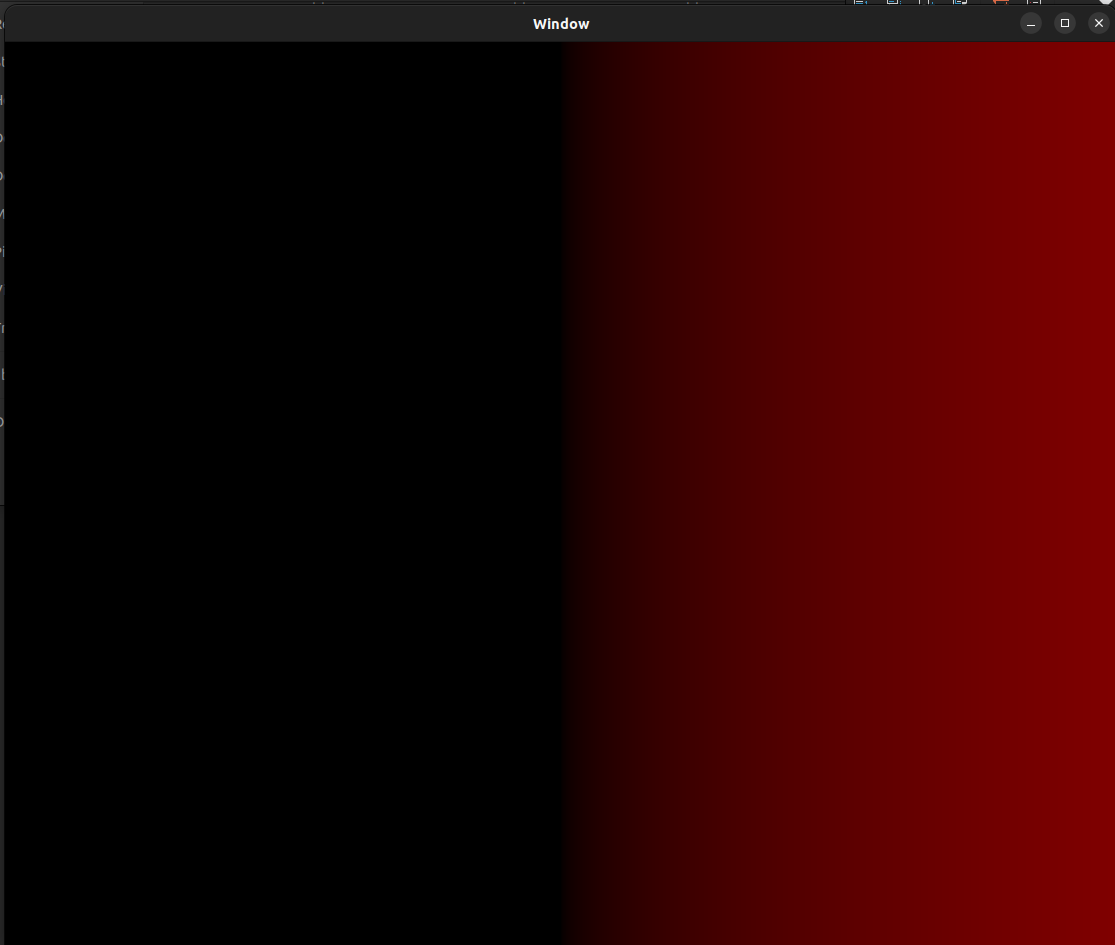
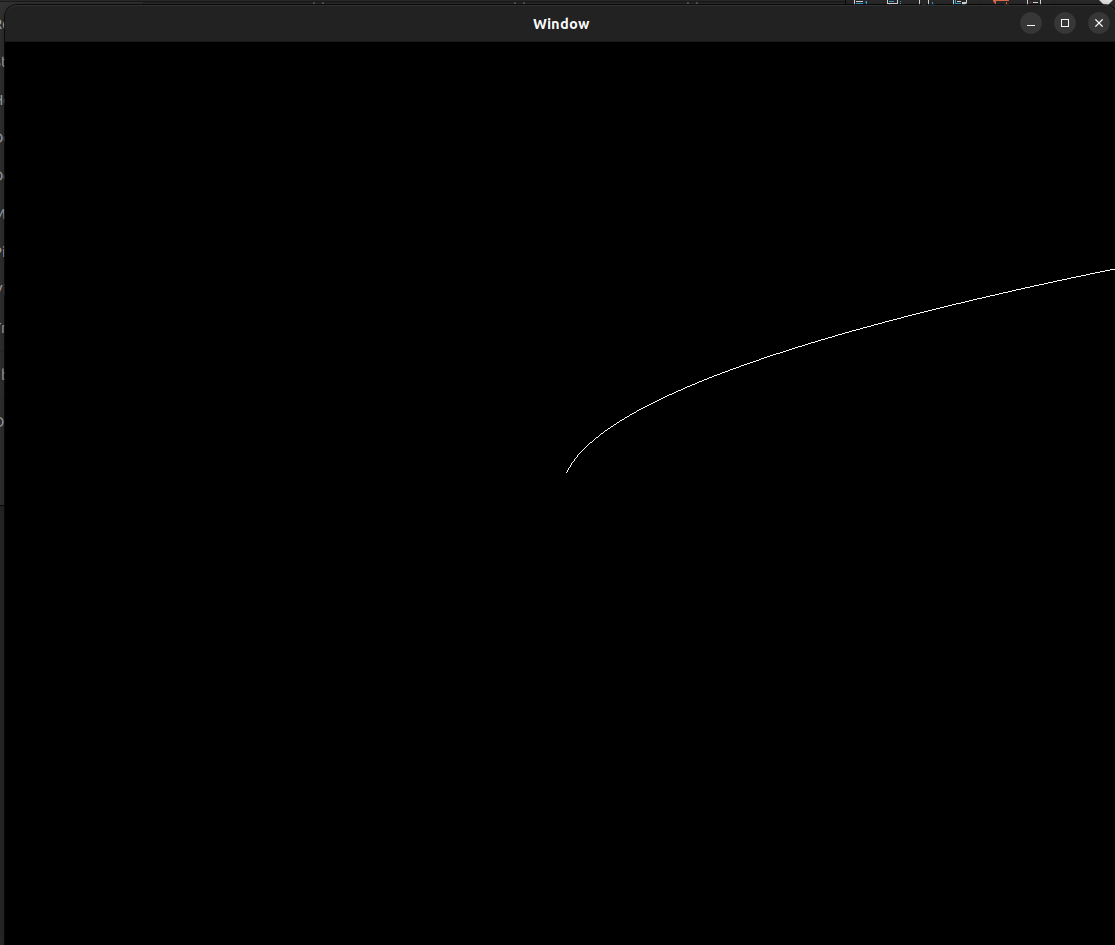


Bu fonksiyonda 1 den basladigi icin asla siyah renk goremeyiz,

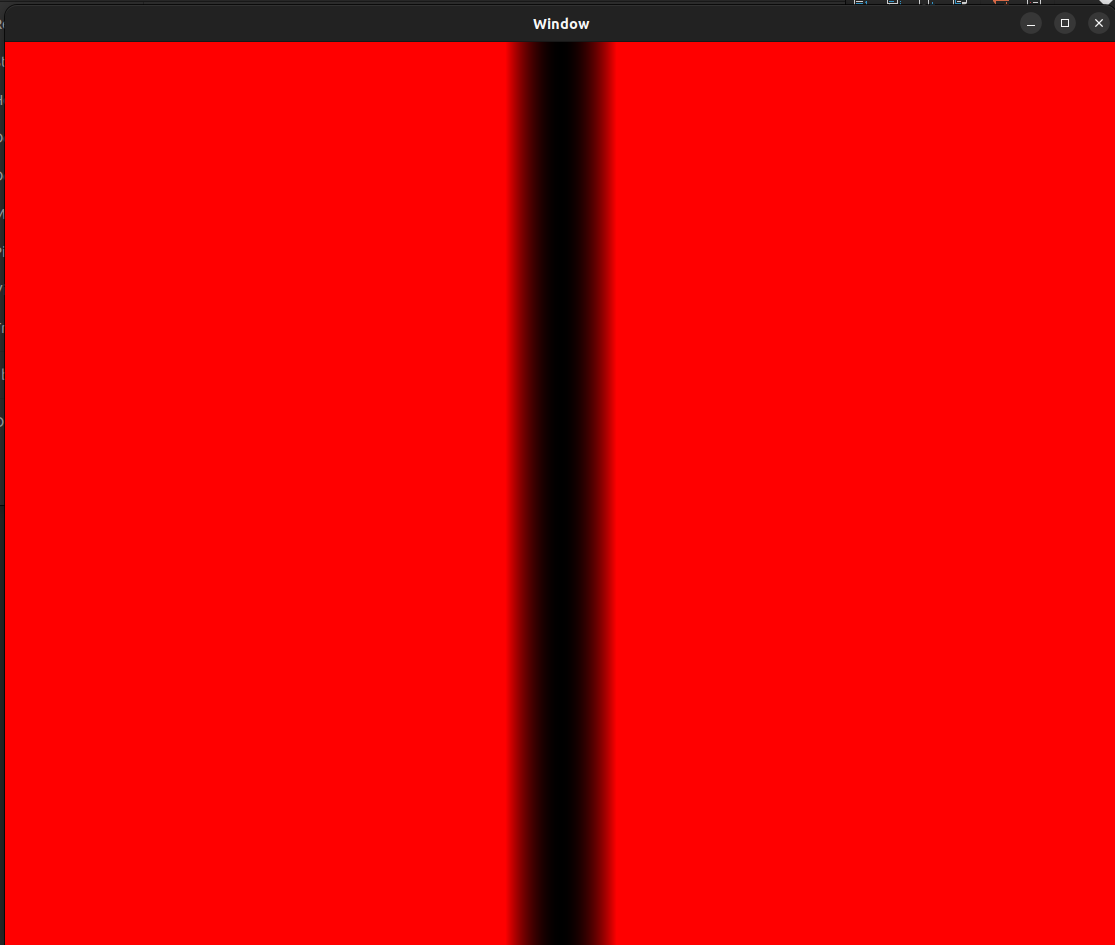
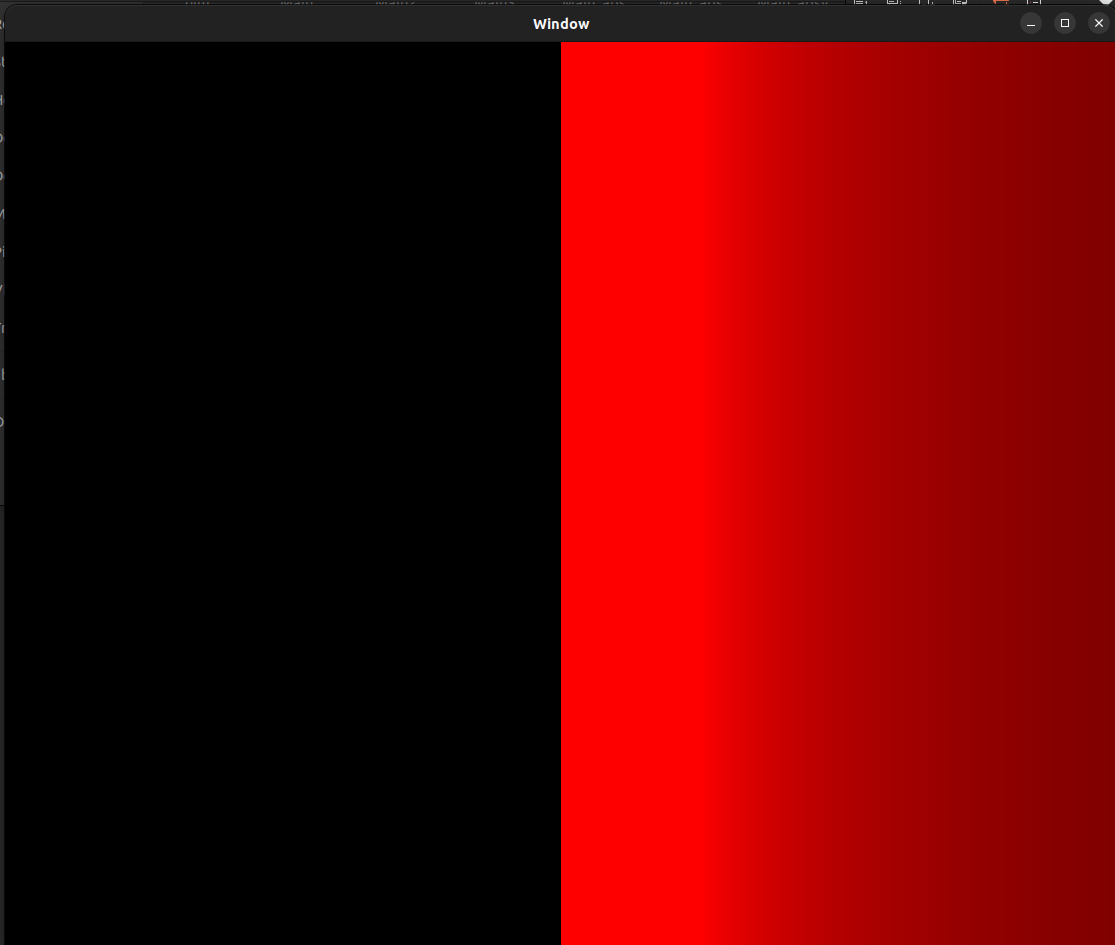
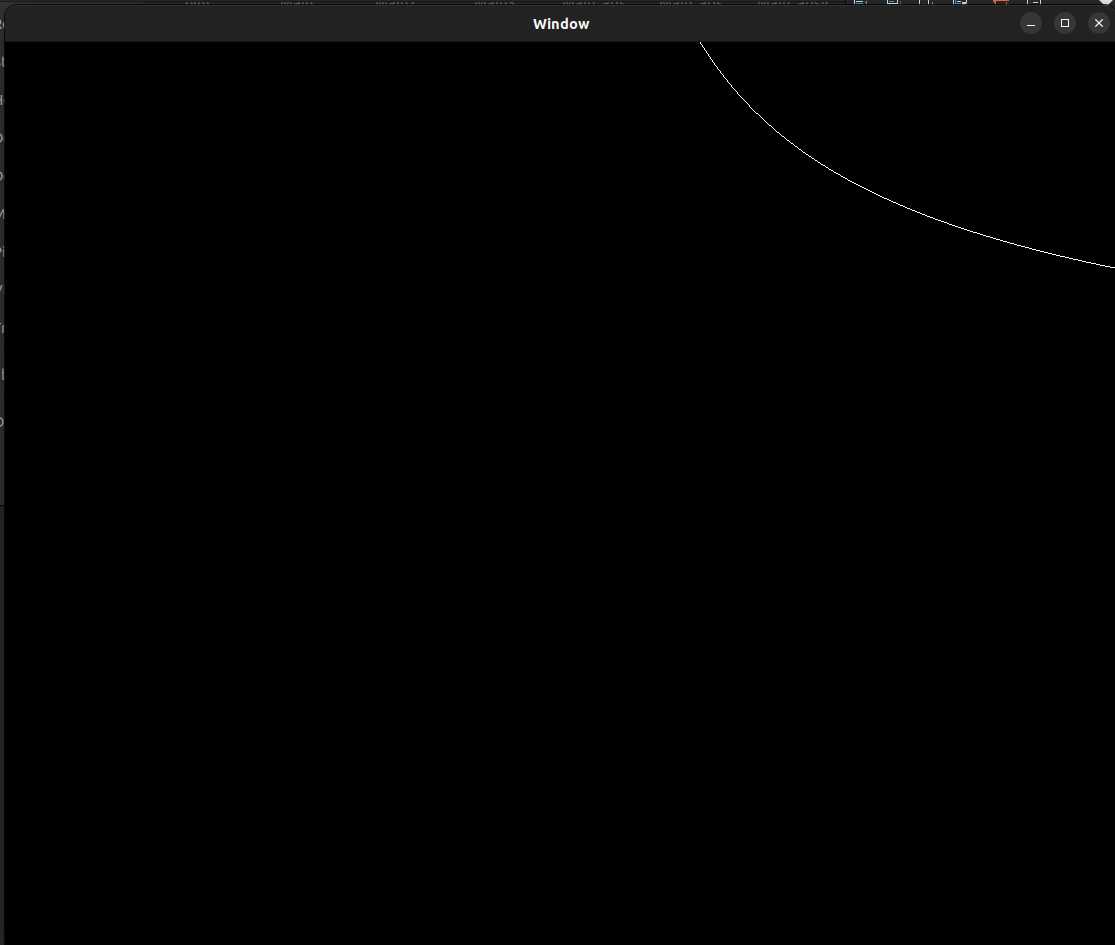
T log2(T x) log2

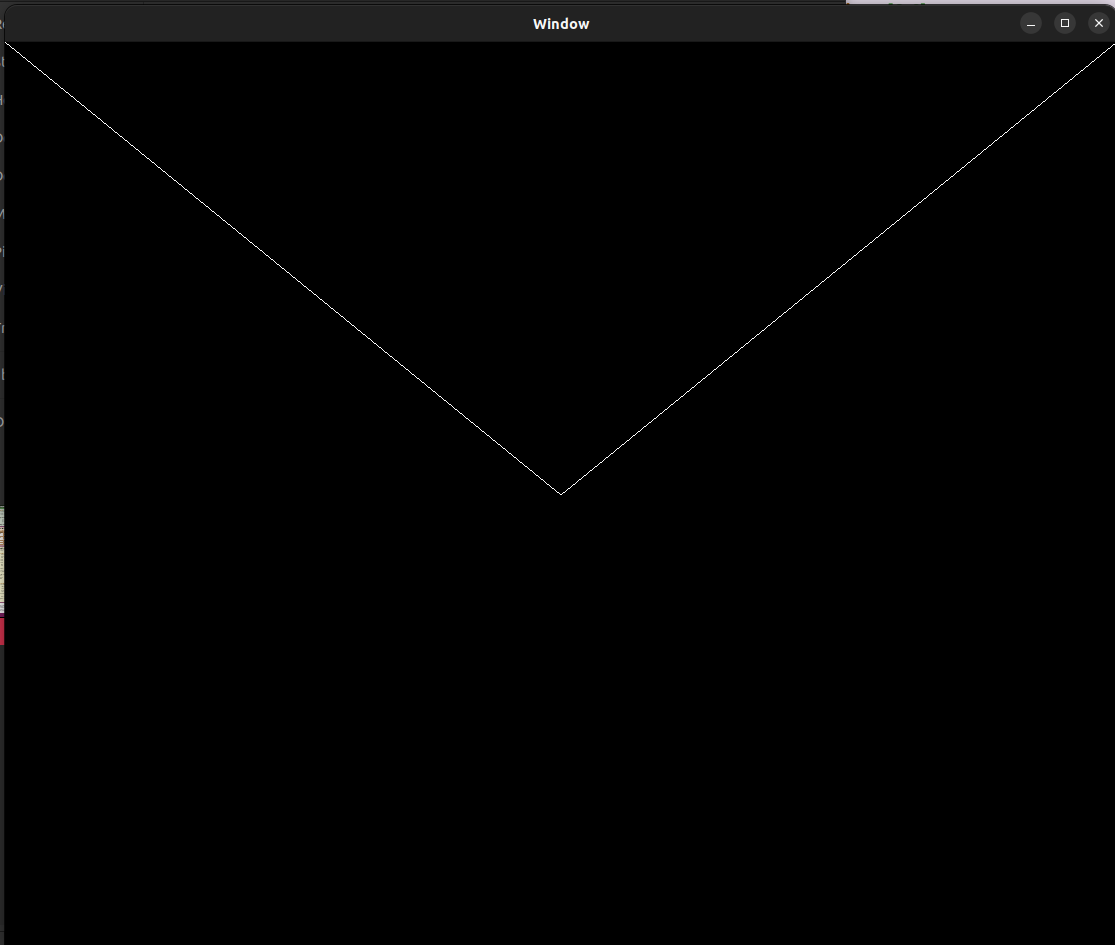


T sqrt(T x) square root

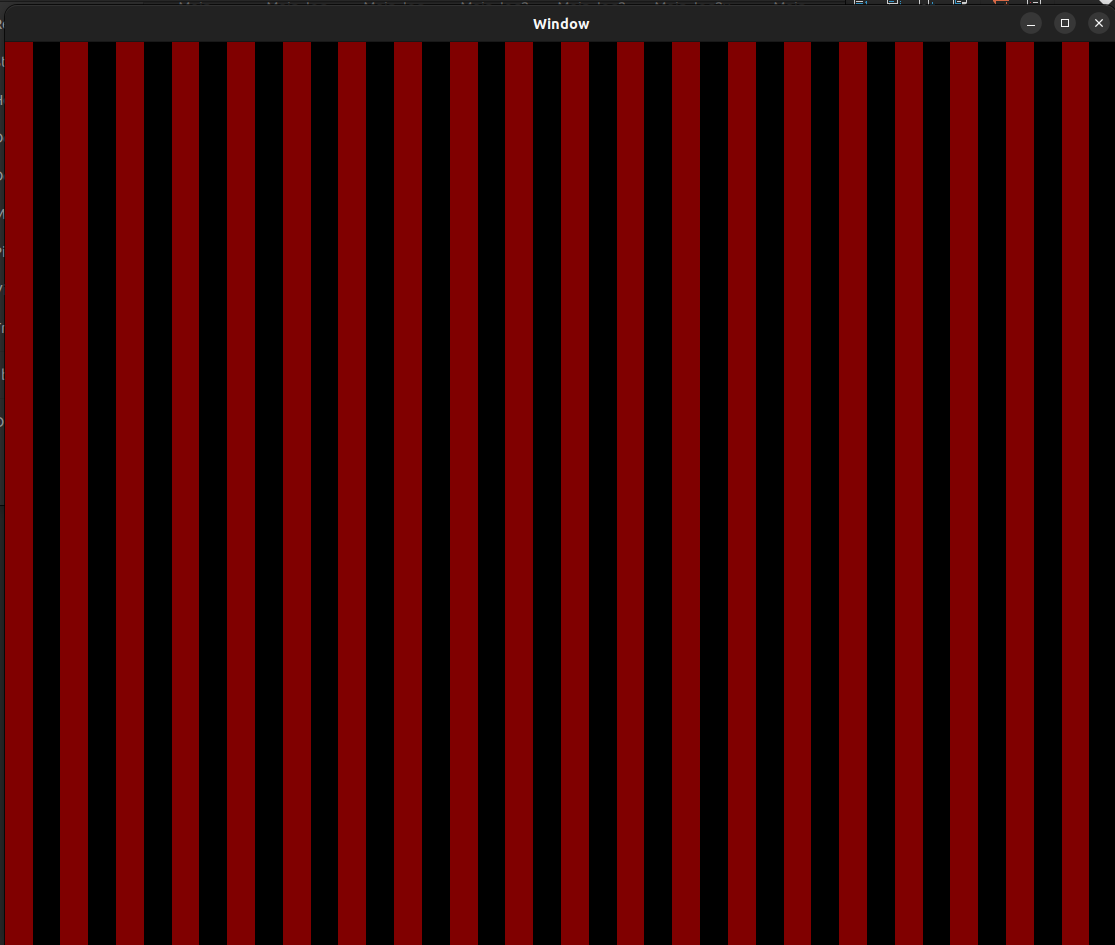
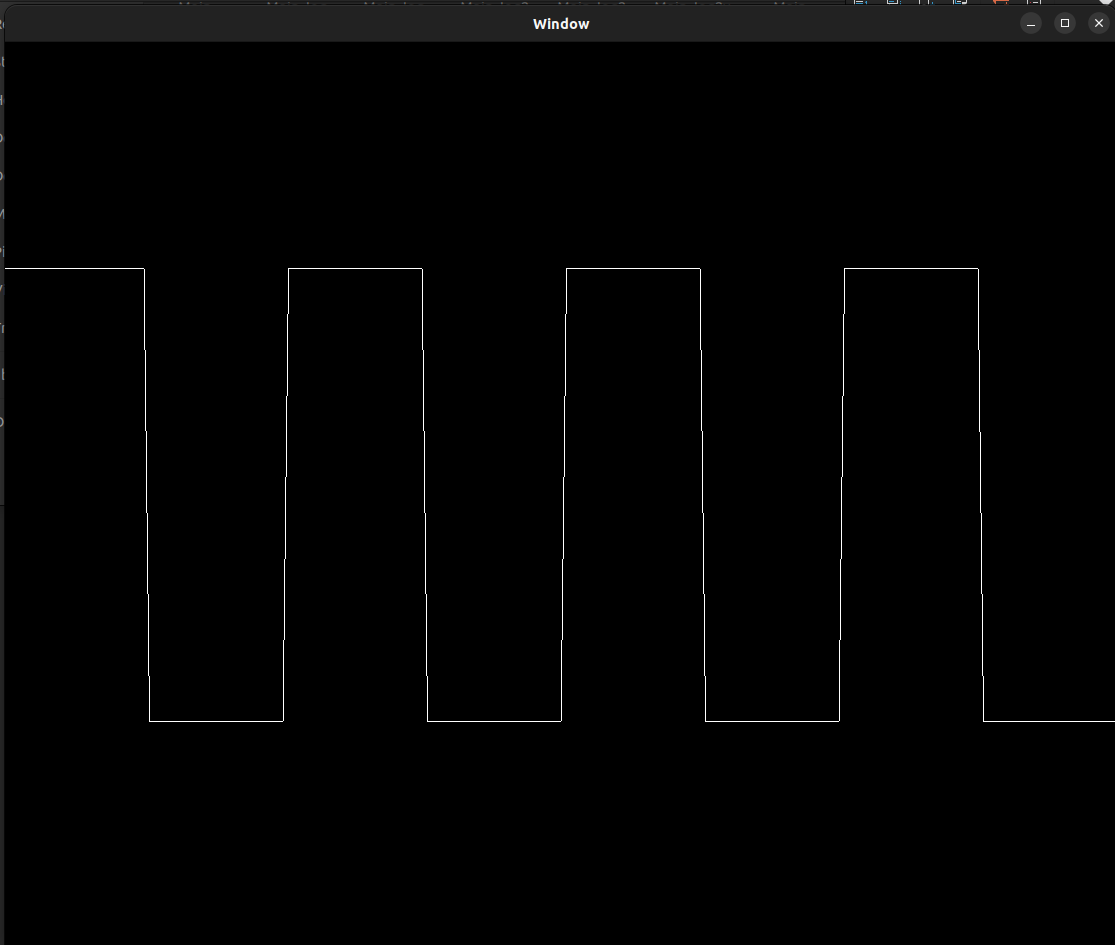


T inversesqrt(T x) inverse square root

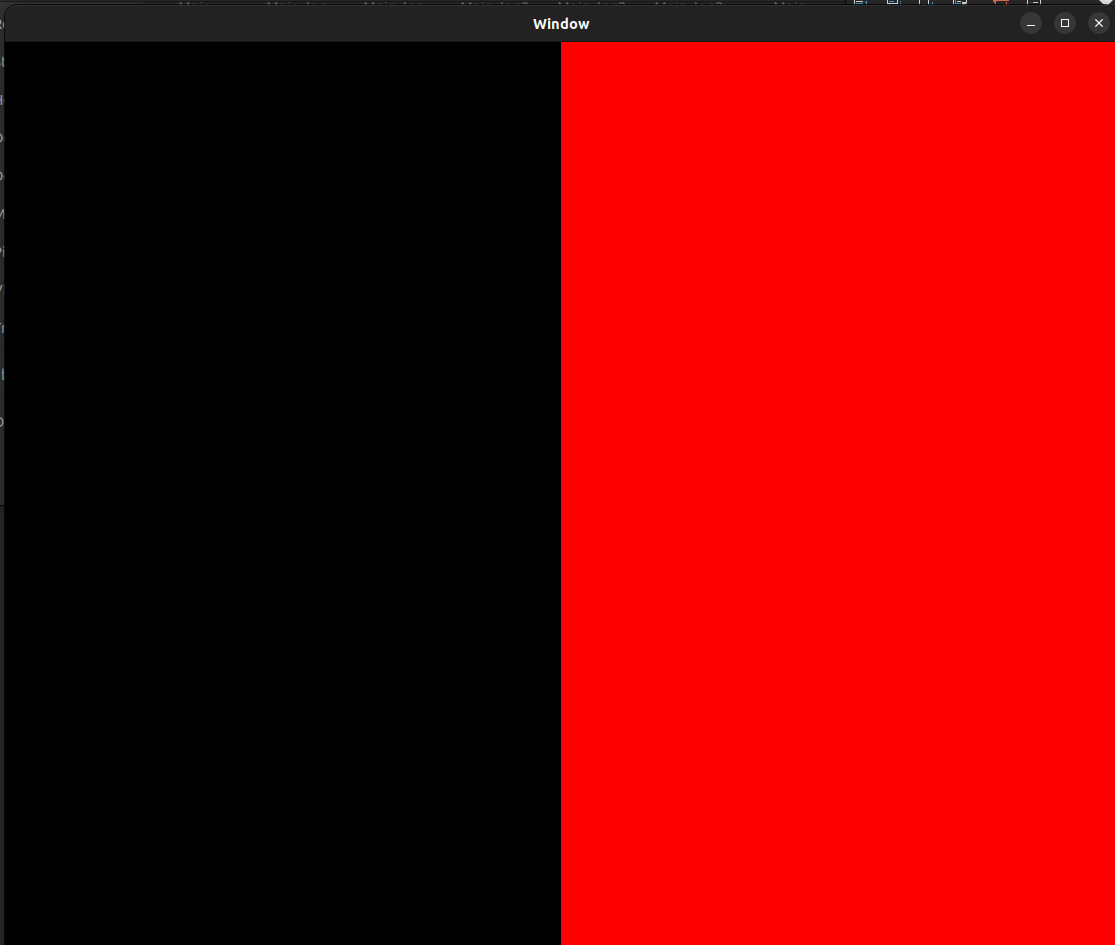
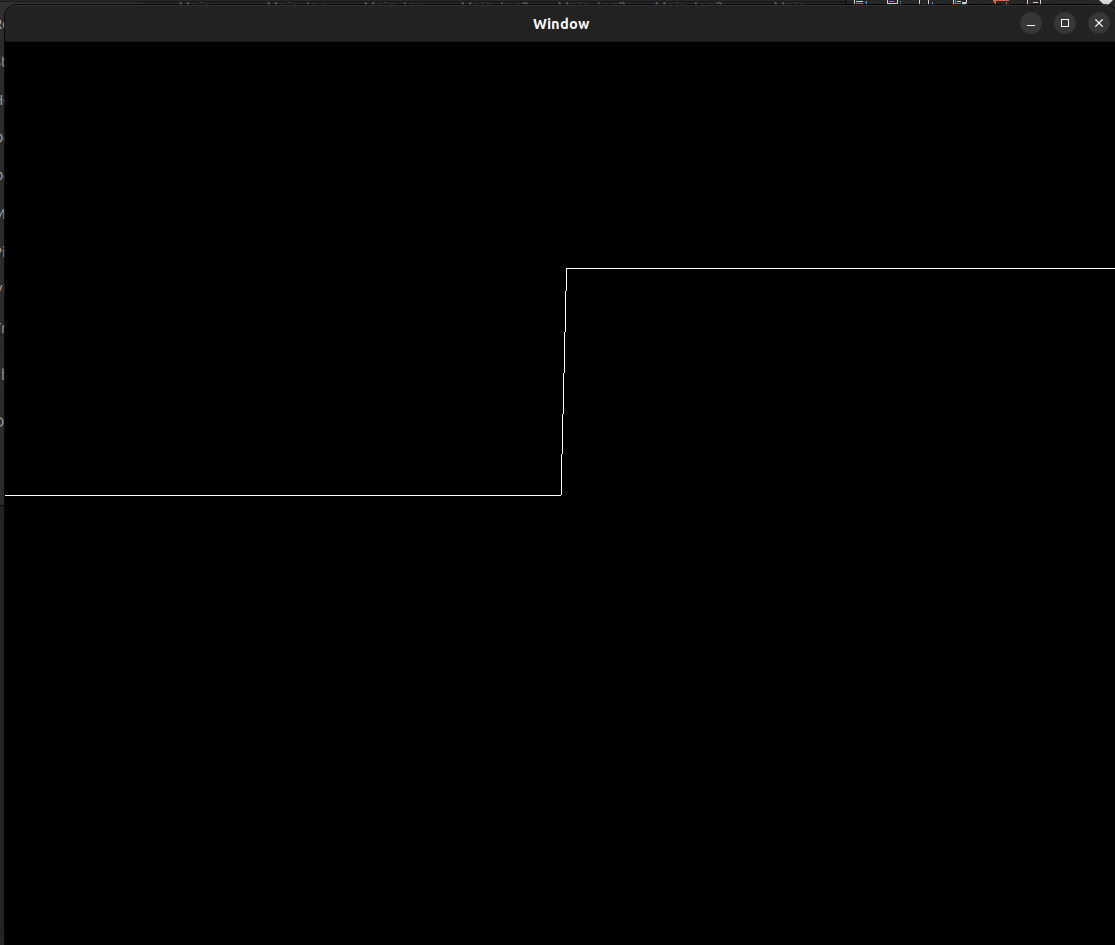
T abs(T x) absolute value



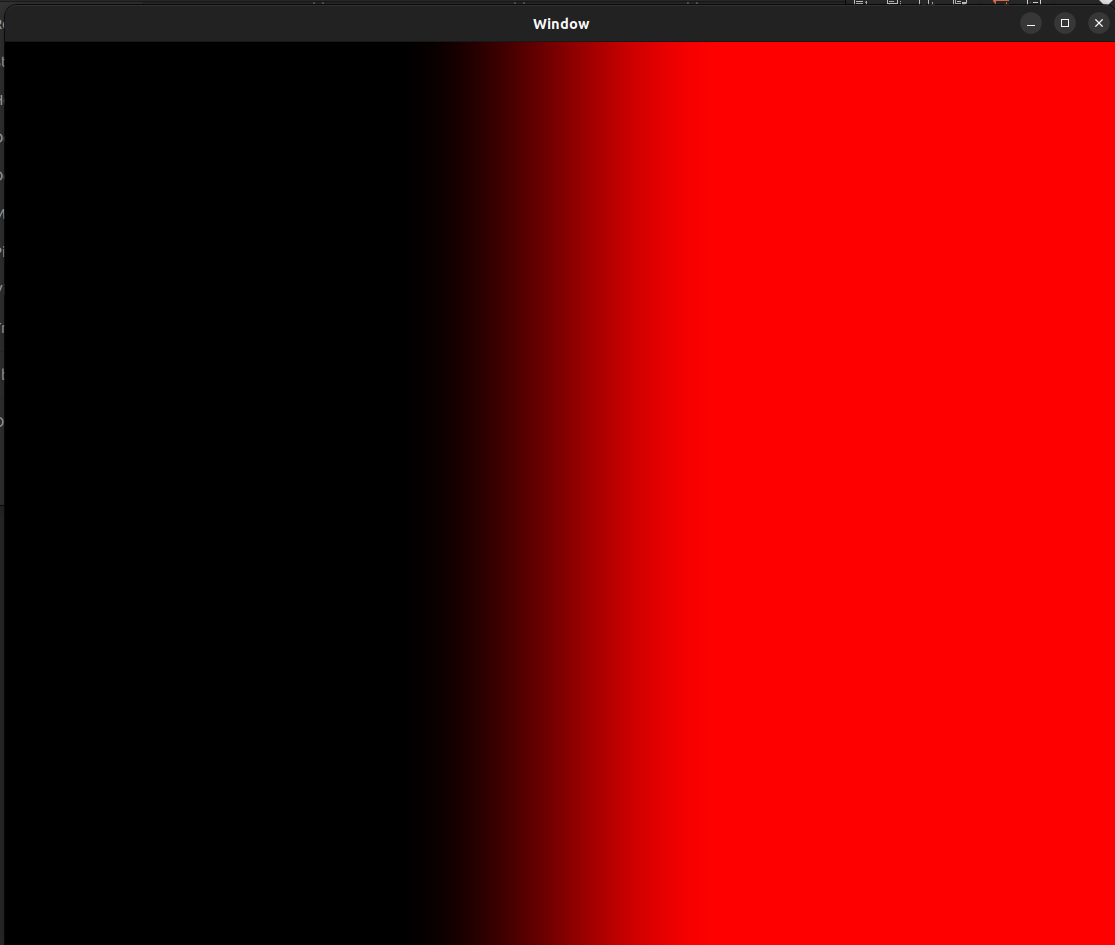
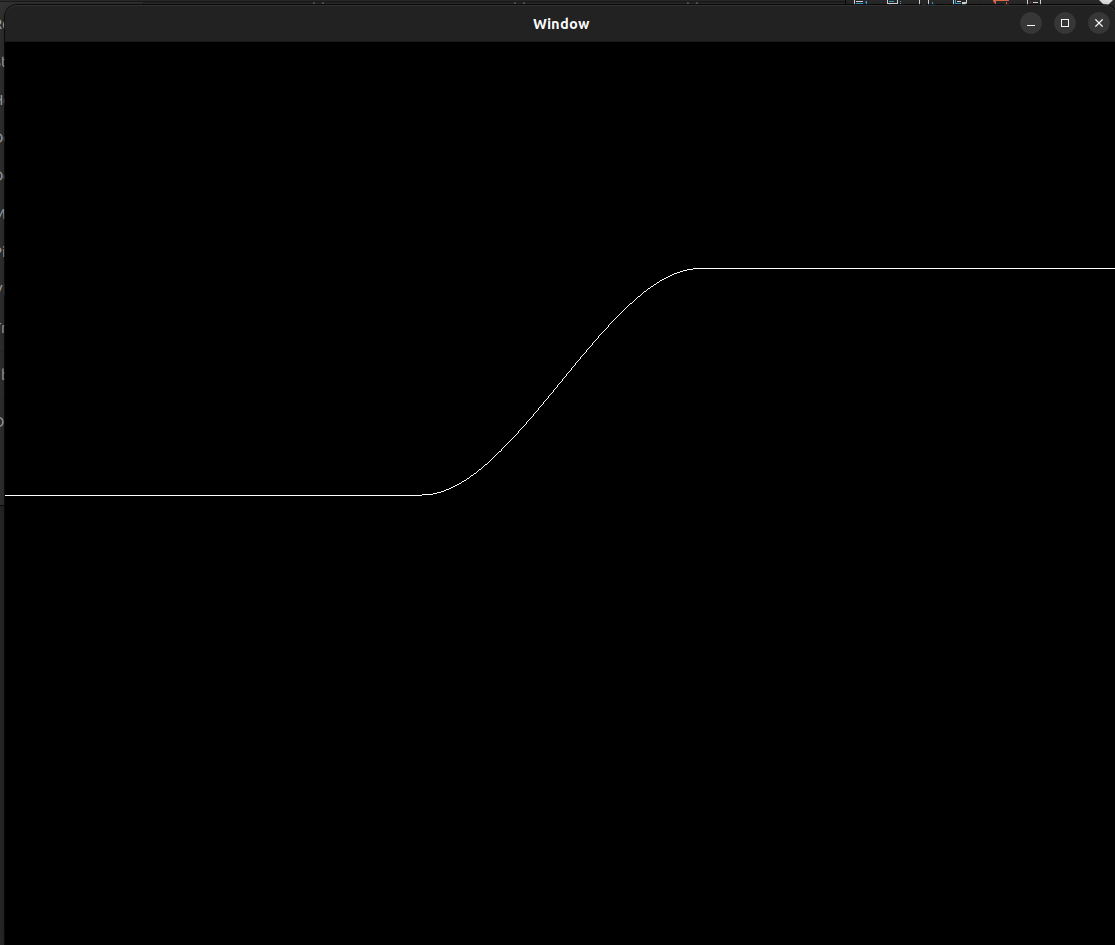
T sign(T x)



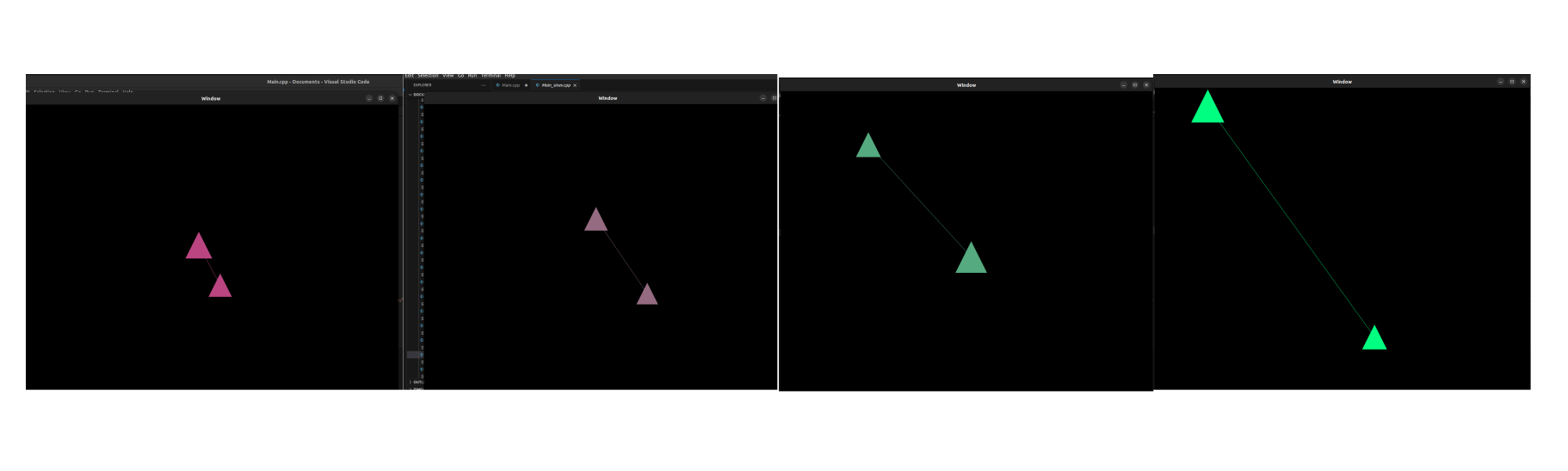
T step(T edge, T x)



T smoothstep(T edge0, T edge1, T x)



float distance(T p0, T p1)  
bu geometric fonksiyon 2 tane vector arasindaki eucledian mesafeyi hesaplamak icin kullanilir. Bunu gosterebilmek icin 2 tane obje cizdim ve bu objeleri cizgi ile birbirine bagladim. Ardindan aralarindaki mesafe arttikca objelerin rengi degistiren bir fragment shader yapmak icin distance fonksiyonunu kullandim.

Goruldugu gibi mesafe arttikca renk degismektedir.

float dist = distance(triangle1Position, triangle2Position);

buradan aralarindaki mesafeyi olcuyoruz ve ardindana FragColordaki degiskenlerin yerine yaziyoruz boylelikle degisken renkli bir cikti elde ediyoruz.

FragColor = vec4(1.0 - dist/2.0, dist/2.0, 0.5, 1.0);